Deformable Simplicial Complexes

Misztal, Marek Krzysztof

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

Deformable Simplicial Complexes

Marek Krzysztof Misztal

Kongens Lyngby 2010
IMM-PHD-2010-241

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

In this dissertation we present a novel method for deformable interface tracking in 2D and 3D—deformable simplicial complexes (DSC). Deformable interfaces are used in several applications, such as fluid simulation, image analysis, reconstruction or structural optimization.

In the DSC method, the interface (curve in 2D; surface in 3D) is represented explicitly as a piecewise linear curve or surface. However, the domain is also subject to discretization: triangulation in 2D; tetrahedralization in 3D. This way, the interface can be alternatively represented as a set of edges/triangles separating triangles/tetrahedra marked as outside from those marked as inside. Such an approach allows for robust topological adaptivity. Among other advantages of the deformable simplicial complexes there are: space adaptivity, ability to handle and preserve sharp features, possibility for topology control. We demonstrate those strengths in several applications.

In particular, a novel, DSC-based fluid dynamics solver has been developed during the PhD project. A special feature of this solver is that due to the fact that DSC maintains an explicit interface representation, surface tension is more easily dealt with.

One particular advantage of DSC is the fact that as an alternative to topology adaptivity, topology control is also possible. This is exploited in the construction of cut loci on tori where a front expands from a single point on a torus and stops when it self-intersects.

Resumé

I denne afhandling beskrives en ny metode til at følge deformerbare grænseflader mellem to stoflige faser i 2D og i 3D. Metoden kaldes for DSC, Deformerbare Simplicial Complexes. Deformerbare grænseflader har mange anvendelser, herunder simulering af væsker, rekonstruktion af overflader, strukturel optimering, billedanalyse, m.m.

I DSC metoden repræsenteres grænsefladen (kurve i 2D, flade i 3D) eksplicit som en stykvist lineær kurve eller stykvist plan flade. Dog er selve domænet som kurven eller fladen befinder sig i også diskretiseret via et simplex net: I 2D er der tale om en triangulering og i 3D en tetrahedralisering. På denne måde kan man alternativt se grænsefladen som en mængde af kanter (trekanter) der adskiller trekanter (tetraeder) som er markeret som havende forskellig fase (som regel er faserne blot "indenfor" og "udenfor"). Denne fremgangsmåde tillader robust topologisk tilpasning. En anden fordel ved DSC er rumlig tilpasning - vi kan have forskellig detaljering af grænsefladen forskellige steder. desuden er det muligt at håndtere skarpe kanter og det er muligt at kontrollere topologien, d.v.s. forhindre topologiske forandringer, hvis dele af grænsefladen støder sammen. Vi demonstrerer disse styrker ved metoden i forbindelse med en række anvendelser.

Specielt præsenteres en ny DSC baseret løser til fluiddynamiske problemer, som er udviklet under PhD projektet. En særlig egenskab ved denne løser er at, fordi DSC bevaerer en eksplicit repræsentation af overfladen, så er det lettere at håndtere overfladespænding end ellers.

En særlig fordel ved DSC er det faktum at man i stedet for at tillade topologiske forandringer kan forhindre dem i at ske. Dette udnyttes i forbindelse med konstruktionen af "cut loci" på en torus, hvor en grænsekurve ekspanderer langs

overfladen på en torus ud fra et enkelt punkt og stopper når kurven rammer sig selv.

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering.

The thesis deals with mathematical modelling using deformable interface tracking methods. The main focus is on introducing a novel, unstructured grid based method, significantly different from the existing deformable models. Several applications are also presented.

The thesis consists of a summary report and a collection of four research papers written during the period October 2007–September 2010, and elsewhere published (or submitted for publication).

Kongens Lyngby, September 2010

Marek Krzysztof Misztal

Papers Included in the Thesis

[3] Marek Krzysztof Misztal, Jakob Andreas Bærentzen, François Anton and Kenny Erleben. Tetrahedral Mesh Improvement Using Multi-face Retriangulation. In *Proceedings of the 18th International Meshing Roundtable*, Salt Lake City 2009. Published.

[4] Marek Krzysztof Misztal and Jakob Andreas Bærentzen. Deformable Simplicial Complexes. Submitted to *ACM Transactions on Graphics*.

[5] Marek Krzysztof Misztal,, Jakob Andreas Bærentzen and Steen Markvorsen. Cut Locus Construction using Deformable Simplicial Complexes. Submitted to *Experimental Mathematics*.

[6] Marek Krzysztof Misztal, Robert Bridson, Kenny Erleben, Jakob Andreas Bærentzen and François Anton. Optimization-based Fluid Simulation on Unstructured Meshes. In *Proceedings of VRIPHYS 2010: The 7th Workshop on Virtual Reality Interaction and Physical Simulation*, Copenhagen 2010. Accepted.

Acknowledgements

This little book would not appear if it was not for my advisor, Jakob Andreas Bærentzen. Andreas came up with the deformable simplicial complexes idea and is the author of the initial, 2D implementation of the method. Besides that, Andreas has been an excellent mentor, always ready to share his insight and good advice and showing an enormous amount of patience.

I would also like to thank my co-advisor François Anton and my frequent collaborator Kenny Erleben (Department of Computer Science, University of Copenhagen) for several good ideas and inspiring conversations.

I could not overestimate the role of Robert Bridson from University of British Columbia, where I spent my external research stay. Robert helped me polish the details of the deformable simplicial complexes method and offered his great insight in fluid dynamics while guiding me through developing a novel fluid solver.

My thanks go to all the researchers whom I had pleasure to meet and who had offered me their advice and help, especially Jeppe Revall Frisvad, Steen Markvorsen, Matthias Stolpe, Carl-Ollivier Gooch and to my colleagues and fellow PhD students, especially Vedrana Andresen, Katarzyna Gębal, Vesselin Perfanov, Ojaswa Sharma, Peter Stanley Jørgensen and Lasse Farnung Laursen for ongoing support and inspiration. I am also grateful to Eina Boeck, the secretary of the IACG group, for her incredible patience while helping me deal with the administrational issues.

Last, but not least, I thank my parents Urszula and Robert Misztal, my sister

Anna and other family members, as well as my friends in Poland, Denmark, Canada and other places for having faith in me and being there for me during the ups and downs of my PhD study.

$\mathbf{p}, \mathbf{u}, \mathbf{v}, \ldots$	points and vectors from $\mathbb{R}^2$ or $\mathbb{R}^3$
$\alpha, \beta, \gamma, \ldots$	scalar values (real numbers), scalar functions
$\mathbf{A}$, $\mathbf{D}$, $\mathbf{S}, \ldots$	matrices
$\mathbf{v}_1, \mathbf{v}_2, \ldots$	vertices (0-simplices)
$e_1, e_2, \ldots$	edges (1-simplices)
$f_1, f_2, \ldots$	faces (2-simplices)
$t_1, t_2, \ldots$	tetrahedra (3-simplices)
$\sigma_1, \sigma_2, \ldots$	arbitrary dimension simplices
σ^d	arbitrary d-simplex
$\langle \mathbf{v}_1, \ldots, \mathbf{v}_{k+1} \rangle$	k-simplex spanned by the vertices $\mathbf{v}_1$, …, $\mathbf{v}_k$
$[\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}]$	oriented k-simplex spanned by the vertices $\mathbf{v}_1$, …, $\mathbf{v}_k$
$\Sigma_1, \Sigma_2, \ldots$	arbitrary, unordered set of simplices
$\lvert\Sigma\rvert$	number of elements in a simplex set Σ
$\mathrm{vert}(\sigma)$	set of vertices of a simplex σ
$\dim(\sigma)$	dimension of a simplex σ
$\dim(\Sigma)$	dimension of a simplex set Σ
$\mathrm{filter}_k(\Sigma)$	k-subset of a simplex set Σ
K	Euclidean simplicial complex
$V(\mathsf{K})$	vertex set of a simplicial complex K
$B_{p,q}(\sigma^p)$	boundary relation of a p-simplex $\sigma^p \in \mathsf{K}$
$C_{p,q}(\sigma^p)$	coboundary relation of a p-simplex $\sigma^p \in \mathsf{K}$
$A_p(\sigma^p)$	adjacency relation of a p-simplex $\sigma^p \in \mathsf{K}$
$\mathrm{st}(\sigma)$	star of a simplex $\sigma \in \mathsf{K}$
$\mathrm{st}(\Sigma)$	star of a simplex set $\Sigma \subset \mathsf{K}$
$\mathrm{cl}(\sigma)$	closure of a simplex $\sigma \in \mathsf{K}$
$\mathrm{cl}(\Sigma)$	closure of a simplex set $\Sigma \in \mathsf{K}$
$\mathrm{lk}(\sigma)$	link of a simplex $\sigma \in \mathsf{K}$

Table 1: Notation used in this document.

Contents

Summary i

Resumé iii

Preface v

Papers Included in the Thesis vii

Acknowledgements ix

1 A Short Introduction to Deformable Interfaces 1
1.1 Deformable Models 5
1.2 Deformable Simplicial Complexes 6

2 Preliminaries 9
2.1 Mathematical Background 10
2.2 Implementation 19

3 Tetrahedral Mesh Improvement Using Multi-face Retriangulation 25
3.1 Introduction and Motivation 26
3.2 Related Work 28
3.3 Tetrahedral Mesh Quality Improvement 31
3.4 Implementation 34
3.5 Tests and Results 35
3.6 Discussion and Future Work 41

4 Deformable Simplicial Complexes 45
4.1 Introduction 46

4.2 Related Works 47
4.3 Deformable Simplicial Complexes 49
4.4 Applications 54
4.5 Conclusions and Future Work 61

5 Cut Locus Construction Using Deformable Simplicial Complexes 63
5.1 Introduction 64
5.2 Method Description 67
5.3 Tests and Results 72
5.4 Discussion 74

6 Optimization-based Fluid Simulation on Unstructured Meshes 77
6.1 Introduction 78
6.2 Related Works on Fluid Solvers 79
6.3 Deformable Interface Tracking 80
6.4 Fluid Simulation 82
6.5 Tests and Results 91
6.6 Conclusions and Future Work 93

7 Conclusions and outlook 95

Chapter 1

A Short Introduction to Deformable Interfaces

We deal with curves and surfaces deforming at various rates in everyday situations (as shown in Figure 1.1). The examples of those include: surface of water, clothes, coastlines, surface of blown glass or modelling clay in the hands of a sculptor. Besides that, for many years we have been using deforming curves and surfaces in a more abstract setting, in order to describe natural phenomena. Those useful abstractions include, for example: progressing wavefronts and contour lines in meteorological maps. However, in this work we restrict our interest to a group of curves and surfaces called *interfaces*. A curve or surface is an interface if it separates a region distinguished by a certain property from

Figure 1.1: Examples of deforming curves and surfaces in everyday life: free surface of water; veils of the Chinese dancers (courtesy of wikipedia.org); contour lines on a weather map (courtesy of Danmarks Meteorologiske Institut).

Figure 1.2: To the left – original volume, middle and right – results of *sculping* using volumetric deformation tools, courtesy of Andreas Bærentzen.

other regions. A water surface is obviously an interface (it separates water from other phases); so are contour lines in maps (they separate regions where the values of a given, continuous quantity is greater or smaller than the value on the contour line); so are wavefronts (they separate region already visited by the wave from the region that the wave has not yet reached). A piece of cloth is not an interface as it does not separate two regions of different properties. We restrict our attention even further to a group of interfaces which are *closed* or *watertight*. This is the case when the region defining the interface is bounded.

It is not surprising that there has been demand for robust and efficient numerical methods for deformable interface tracking (*deformable models*), as we both want to represent familiar deformable objects in virtual environments, as well as use them as a description method in more abstract applications. First deformable models began to appear in late 1980ties (*active contours* [46] in 1987, *level set method* [74] in 1988). Research in this field has been booming ever since, as increasing computational power opened doors for new applications and improvement of the interface tracking algorithms. Several excellent methods have been developed, often custom tailored for the requirements of specific areas of use.

Geometric modelling. The interface is the boundary between modelling material and empty space. While traditional geometric modelling methods do not normally employ deformable models per se, use of deformable interfaces in more recent techniques, such as *sculpting* [6] (see Figure 1.2) allowed for developing more intuitive interfaces. Deformable models used in sculpting should handle changes in the topology of an interface robustly, allow details of different magnitude and sharp details. Another property, crucial in this application, is interactivity – the chosen method should not introduce too much time overhead.

Image analysis. One of the important tasks in 2D and 3D image analysis is

Figure 1.3: Segmentation of the CT scan of a human thoracic cage, courtesy of Ojaswa Sharma [84].

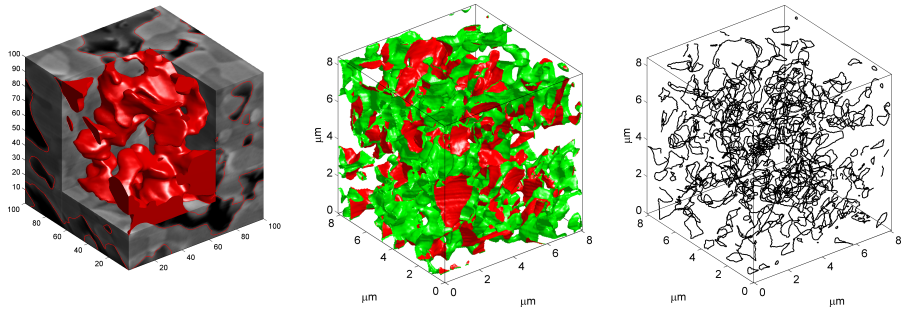

Figure 1.4: Results of segmentation of the porosity in a Solid Oxide Fuel Cell sample, performed in order to localize the triple phase (pore, nickel and YSZ) boundaries in the structure. Courtesy of Peter Stanley Jørgensen [45].

segmentation – partitioning the image into multiple, meaningful *segments* (e.g. by labelling the pixels), in order to localize objects within the image (as shown in Figures 1.3, 1.4, see also [14]). This can be done using deformable models, especially when the images are noisy and using traditional clustering methods would lead to semantically wrong results (e.g. by creating too many components). Topological adaptivity is often desired, although in some applications, when we have prior knowledge about the topology of the segments, we might want to preserve it – a classic and challenging example is reconstruction of the cortical surface of a brain, which is topologically equivalent to a sphere [8]. Another important application of deformable models in image analysis are in image restoration (*denoising*, *deblurring*) [63, 81]

Reconstruction. One step further from segmentation is reconstruction of objects from images or discrete point-sets (e.g. [102]). The interface naturally separates the reconstructed object from its exterior. Topological adaptivity might be desired, but if we have prior knowledge about the topology of the object, we might want to preserve it. We also expect deformable model to handle details of different magnitudes and, in some cases, to handle sharp features properly.

Fluid simulations. In fluid simulations, deformable models are normally used to track contact surface between two fluids (e.g.: water and air, smoke and air, etc. [11]). Robust topological adaptivity is required, since the volume of water tends to split and merge with ease, while sharp detail preservation is of lesser importance (although space adaptivity is often needed). In case of pre-rendered simulations time restrictions are lenient, but in case of real-time simulations time efficiency is crucial. In some applications we might also want to have direct access to the surface parametrization (e.g. showing the effects of significant surface tension; properly handling solid boundaries). We present our own, novel fluid solver in Chapter 6.

Structural optimization. There are two main sub-problems in structural optimization: topology optimization and shape optimization [7,56]. In the former, an optimal distribution of material in the design domain is sought (so that e.g. the strength of the structure is maximized). In the latter, optimal shape of the structure boundary is found, without changing its topology. Deformable models can be used in both stages of optimization and in both cases the interface separates material from the empty space. In the first stage, robust topological adaptivity is important [94]. In the second stage – space adaptivity, explicit interface representation and robust handling of sharp features.

Summarizing, depending on an application, we might be interested in the following properties of deformable models:

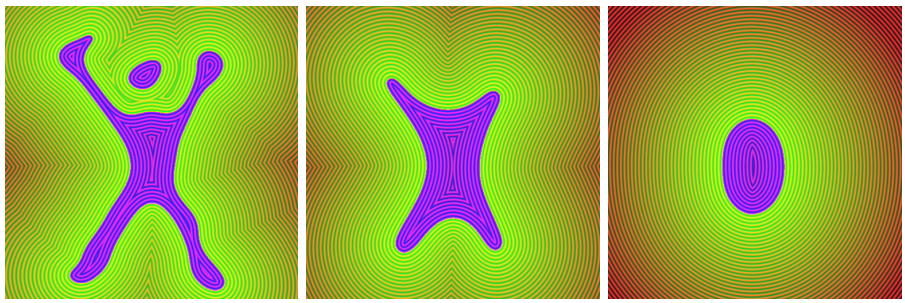

Figure 1.5: Interface evolution in the level set method (courtesy of Andreas Bærentzen).

- **topological adaptivity** – ability of the method to automatically change the topology of the interface (merge or split parts of the interface), whenever collision occurs;
- **topology control** – ability of the method to *preserve* the topology of the interface;
- **scale adaptivity** – ability of the method to efficiently and accurately represent details of different magnitude;
- **sharp detail preservation** – some deformable models introduce significant *numerical diffusion* leading to smoothing out sharp details. In some cases the representation alone is not able to capture sharp details (e.g. voxel grid);
- **availability of explicit interface representation**;
- **interactivity**;

1.1 Deformable Models

Traditionally, deformable interface tracking methods, also known as *deformable models*, fall into two groups: *explicit* (or *Lagrangian*) and *implicit* (or *Eulerian*). Lagrangian methods, such as *snakes*, use parametrisation of the interface and apply the deforming velocity field ($\mathbf{u}$) directly to the interface points ($\mathbf{v}$):

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \mathbf{u}(\mathbf{p}).$$

This approach is effective for small deformations, but leads to trouble when parts of the interface collide and the topology of the interface changes. Reparametrisation is needed, along with surgical cuts and efficient collision detection mechanism. Those problems do not occur in Eulerian methods, such as the *level set method* (*LSM*, Figure 1.5, for detailed description of the method see [75, 83]). LSM represents an interface as the 0-level set of a function ϕ defined over the whole space. The deformation of an interface is produced by evolving *the function* ϕ. Such an evolution is rather non-trivial: if we want to produce the deformation of the interface due to the veolocity $\mathbf{u}$, the evolution of the function ϕ is given by the following partial differential equation:

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = 0,$$

also known as *level set equation*. What we earn this way is trivial and robust topological adaptivity, as the changes in the topology of the 0-level set happen automatically. The *signed distance function* (the shortest distance from the interface, positive outside the interface and negative inside the interface) is the usual choice for ϕ. One can easily notice how simple it is to produce the *interface offsetting* – motion in the normal direction – in this setting. If the interface is given as the 0-level set of a signed distance function ϕ, its offset produced by moving in the normal direction at unit speed in time τ is simply the 0-level set of a function $\phi_\tau = \phi - \tau$.

However, the LSM also exhibits several drawbacks: it is bound to a certain scale dictated by the resolution of the underlying grid; it suffers from significant numerical diffusion due to the constant re-sampling of the interface and also the properties of the numerical methods used to solve the level set equation [1]; it does not allow explicit interface representation, but the interface has to be reconstructed at every time step using an implicit surface polygonization algorithm such as *marching cubes* [60, 72]. These shortcomings could be avoided while keeping robust topological adaptivity by using deformable simplicial complexes.

A more detailed description of the state-of-the-art in the field of deformable models can be found in Chapter 4.

1.2 Deformable Simplicial Complexes

Deformable simplicial complexes (*DSC*) represent the interface in an explicit, discretised way – as a piecewise linear curve in 2D, or a piecewise linear (trian-

[1] For more information on the properties of Hamilton-Jacobi equations (such as level set equation) and grid-based finite difference methods for solving them see [26, 58].

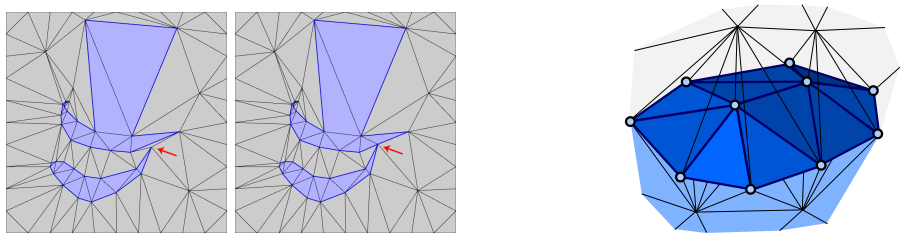

Figure 1.6: Interface representation in deformable simplicial complexes (2D on the left, 3D on the right). Exterior triangles (tetrahedra) are light gray, interior – blue. Simplices belonging to the interface (edges and vertices in 2D; faces, edges and vertices in 3D) are highlighted in dark blue. On the left, the red arrow indicates where topology changes take place. Note also the difference in scale between the largest and the smallest triangles.

gulated) surface in 3D. However, also the whole embedding space (or rather, an interesting part of it, which is usually a box containing the interface) is a subject to a discretization – triangulation in 2D or tetrahedralization in 3D, fulfilling two conditions:

1. *Simplicial complex criterion* – the intersection of two simplices can be either empty or be their common face (e.g. in 3D two tetrahedra can intersect only at the common face, edge or vertex; in 2D two triangles can intersect only at the common edge or vertex);

2. It conforms to the interface – the interface is represented as a set of boundary edges (faces) between the triangles (tetrahedra) marked as *outside* and *inside*;

In other words, the whole domain is divided into simplices (triangles in 2D, tetrahedra in 3D): *interior* and *exterior* (see Figure 1.6) in such a way, that the interface is given as a set of simplex boundary faces (edges in 2D; triangles in 3D) dividing the interior from the exterior. This implies, that the interface is in fact given implicitly and the method is oblivious of the parametrisation of the interface. The interface deformation is performed via displacements of the interface vertices, while keeping the simplicial complex criterion of the underlying triangulation (tetrahedralization) all the time.

As vertices move, some of the triangles (tetrahedra) become degenerate and are simply removed by DSC's mesh quality improvement procedures. For instance, if a vertex from one component of the interface comes close to another component, a very degenerate triangle appears (as exemplified in 1.6). This triangle is

removed by edge flipping which causes the two components to merge. In fact all interface collisions are detected and resolved this way – locally – on individual triangle (tetrahedron) level. That means, there is no external collision detection mechanism needed in this framework, and the changes in the topology of the interface are automatic, transparent to the user.

This approach allows DSC share the biggest advantage of the level set method:

- robust topological adaptivity;

while preserving the advantages of the explicit methods:

- little numerical diffusion;
- availability of the interface parametrisation;

Moreover, use of an unstructured grid yields scale adaptivity, and intrinsic collision detection mechanism allows for topology control.

Chapter 2

Preliminaries

In the first part of this chapter we introduce, rather formally, main mathematical notions necessary to understand the implementation of the method. Knowing the implementation details is not necessary to understand the following chapters, however some of the notions defined in Section 2.1.1 are used throughout the document (e.g.: *simplex*, *simplicial complex*, *closure*, *star*, *link*). Simplicial complexes are among the most important concepts used in algebraic topology, so well structured, in-depth description of the subject can be found in many algebraic topology textbooks (such as [57]).

In the second part of this chapter we briefly describe the implementation of the 3D deformable simplicial complexes. We introduce *incidence simplicial* data structure for storing and traveral of simplicial complexes [30] and describe the alterations we have made to the original structure.

2.1 Mathematical Background

2.1.1 Euclidean simplicial complexes

Let $\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}$ be points in a n-dimensional Euclidean space E^n. We call them *affinely dependent* if

$$(\exists \mu_1, \ldots, \mu_{k+1} \in \mathbb{R}) \sum_{i=1}^{k+1} \mu_i = 1 \wedge \sum_{i=1}^{k+1} \mu_i \mathbf{v}_i = 0.$$

Otherwise, $\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}$ are called *affinely independent*.

Having $p+1$ affinely independent points $\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}$ in a E^n we define an *Euclidean simplex* $\sigma = \langle \mathbf{v}_1, \ldots, \mathbf{v}_{p+1} \rangle$ as a set of points given by a formula:

$$\mathbf{v} = \alpha_1 \mathbf{v}_1 + \ldots + \alpha_{p+1} \mathbf{v}_{p+1},$$

where $\alpha_i \geq 0$, $\alpha_1 + \ldots + \alpha_{p+1} = 1$ (in other words, σ is a *convex hull* of points $\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}$). A simplex defined this way is, of course, a closed set in E^n. We say that σ is of *dimension* p or, equivalently, that σ is an *Euclidean p-simplex*. We denote this fact by: $\dim(\sigma) = p$. We can also write the following relation:

$$\dim(\sigma) = |\mathrm{vert}(\sigma)| - 1,$$

where $\mathrm{vert}(\sigma) = \{\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}\}$ is the undordered set of the vertices of σ.

We call a 0-simplex a *vertex*, a 1-simplex an *edge*, a 2-simplex a *face* and a 3-simplex a *tetrahedron*. Further, we call each point $\mathbf{v}_i$ a *vertex* of σ, and each simplex $\langle \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_q} \rangle$ $(0 < q \leq p,\ 1 \leq i_k \leq p)$ a *q-face* of σ (or simply a *face* of σ, if no ambiguity arises). We also call the $(p-1)$-faces of a p-simplex σ^p its *boundary faces*. The faces of σ that are not equal to σ itself are called its *proper faces*. The union of all the boundary faces of a simplex σ is called the *boundary* of σ. The boundary of a p-simplex in E^n should not be confused with the topological notion of the boundary of set A defined as

$$\partial A = \overline{A} \cap \overline{E^n - A}.$$

If $n > p$, $\partial \sigma^p = \sigma^p$. The boundary of a p-simplex should be interpreted in sense of p-manifold boundary.

The *relative interior* of a simplex σ (or the *open simplex* spanned by the points $\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}$) is defined as a set of points:

$$\mathbf{v} = \beta_1 \mathbf{v}_1 + \ldots + \beta_{p+1} \mathbf{v}_{p+1},$$

where $\beta_i > 0$, $\beta_1 + \ldots + \beta_{p+1} = 1$. An open simplex is an open set within its *affine hull*. The affine hull of $p+1$ affinely independent points $\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}$ is defined as a set of points:

$$\mathbf{v} = \gamma_1 \mathbf{v}_1 + \ldots + \gamma_{p+1} \mathbf{v}_{p+1},$$

where $\gamma_1 + \ldots + \gamma_{p+1} = 1$. In E^3, the affine hull of four affinely independent points is the whole space, the affine hull of three affinely independent points is a plane that passes through those points, the affine hull of two affinely independent points is a straight line that passes through those points, and the affine hull of one point is a set containing only this point. It can now be noticed that, although in E^3 an open 2-simplex is not an open set, it is open in the plane that contains it (which is a 2-dimensional affine space). Similarly, an open 1-simplex is an open set in the straight line that contains it (which is a 1-dimensional affine space). Lastly, an open 0-simplex, which is a single point, is an open set within the topological space whose only element is this point (since for every topological space X, X and $\emptyset$ are always open sets).

Clearly, every simplex is a disjoint union of the relative interiors of its faces.

For arbitrary, finite set of simplices Σ we define its *dimension*, as the maximum dimension of the simplices in Σ:

$$\dim(\Sigma) = \max\{\dim(\sigma) : \sigma \in \Sigma\}.$$

We also define a *k-subset* of Σ as a set of all k-simplices in Σ:

$$\text{filter}_k(\Sigma) = \{\sigma_i \in \Sigma : \dim(\sigma_i) = k\}.$$

A finite set Σ of Euclidean simplices forms a (finite) *Euclidean simplicial complex* (see Figure 2.1) if the following two conditions hold:

1. Σ is closed (which means that for each simplex $\sigma \in \Sigma$, all faces of σ are in Σ),
2. The intersection $\sigma_i \cap \sigma_j$ of any two simplices $\sigma_i, \sigma_j \in \Sigma$ is a face of both σ_i and σ_j.

From this point we denote simplicial complexes with sans-serif capital Latin letters, e.g.: K. Any subset $\mathsf{K}' \subset \mathsf{K}$ that is itself a simplicial complex is called a *subcomplex* of K. In particular, for any nonnegative integer k, the subset $\mathsf{K}^{(k)} \subset \mathsf{K}$ consisting of all simplices of dimension less than or equal to k is a subcomplex, called the *k-skeleton* of K. The 0-skeleton of K is called a *vertex set* of K and denoted $V(\mathsf{K})$.

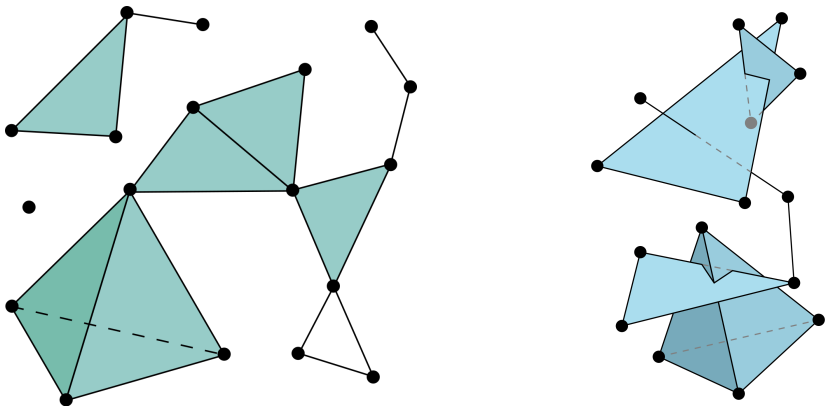

Figure 2.1: An example of what is (right) and what is *not* (left) a simplicial complex. Courtesy of wikipedia.org.

For a p-simplex σ^p in a simplicial complex K we define the following *topological relations*:

- for $p > q$, the *boundary relation* $B_{p,q}(\sigma^p)$ is the set of all q-faces of σ^p:
$$B_{p,q}(\sigma^p) = \text{filter}_q\{\sigma \in \mathsf{K} : \text{vert}(\sigma) \subset \text{vert}(\sigma^p)\},$$
- for $p < q$, the *coboundary relation* $C_{p,q}(\sigma^p)$ is the set of all q-simplices that have σ^p as a face:
$$C_{p,q}(\sigma^p) = \text{filter}_q\{\sigma \in \mathsf{K} : \text{vert}(\sigma^p) \subset \text{vert}(\sigma)\},$$
- for $p > 0$, the *adjacency relation* $A_p(\sigma^p)$ is the set of all p-simplices, which are $(p-1)$-adjacent to σ^p (which means those simplices, that share a $(p-1)$-face with σ^p):
$$A_p(\sigma^p) = \text{filter}_p\{\sigma \in \mathsf{K} : |\text{vert}(\sigma^p) \cap \text{vert}(\sigma)| = p\},$$
- the *adjacency relation* $A_0(\mathbf{v})$, where $\mathbf{v}$ is a vertex, is the set of all vertices $\tilde{\mathbf{v}}$ connected to $\mathbf{v}$ by an edge:
$$A_0(\mathbf{v}) = \{\tilde{\mathbf{v}} \in \text{vert}(\mathsf{K}) : (\exists e \in \mathsf{K}) \quad \text{vert}(e) = \{\mathbf{v}, \tilde{\mathbf{v}}\}\}.$$

We define the *star* of a simplex σ (see Figure 2.2, top) as a set of all the simplices in K, which have σ as a face:

$$\text{st}(\sigma^p) = \{\sigma \in \mathsf{K} : \text{vert}(\sigma^p) \subset \text{vert}(\sigma)\} = \bigcup_{q=p+1}^{n} C_{p,q}(\sigma^p).$$

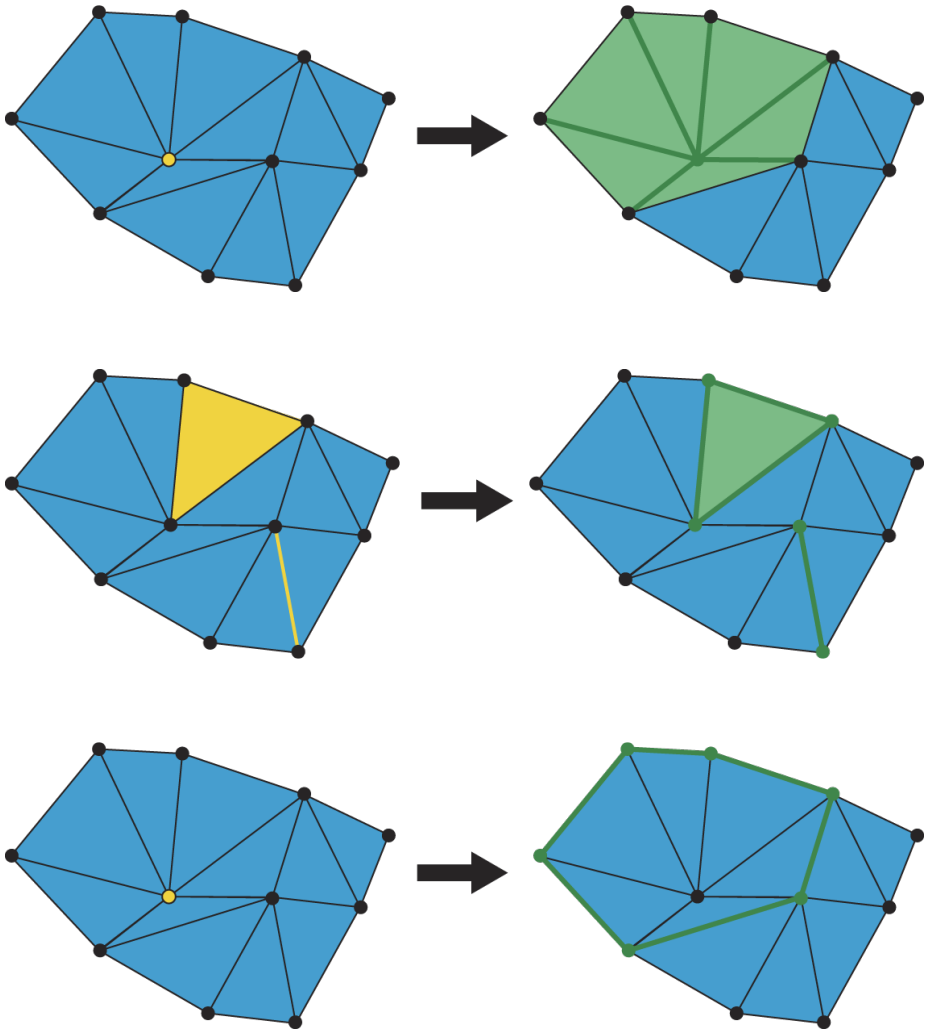

Figure 2.2: Illustration of the notions of *star* (top), *closure* (middle) and *link* (bottom) of a simplex or set of simplices in a simplicial complex. Courtesy of wikipedia.org.

For the sake of convenience, we also define a star of an arbitrary subset Σ of K, as the union of the stars of all simplices in Σ:

$$\mathrm{st}(\Sigma) = \bigcup_{\sigma_i \in \Sigma} \mathrm{st}(\sigma_i).$$

We also define the *closure* (see Figure 2.2, middle) of a simplex $\sigma^p \in \mathsf{K}$ as a set

$$\mathrm{cl}(\sigma^p) = \bigcup_{q=0}^{p} B_{p,q}(\sigma^p)$$

and the closure of a simplex set $\Sigma \subset \mathsf{K}$, expressed as a set

$$\mathrm{cl}(\Sigma) = \bigcup_{\sigma_i \in \Sigma} \mathrm{cl}(\sigma_i).$$

Notice that this *combinatorial* notion of closure differs from the topological notion of closure. In terms of point-set topology, for any simplex σ

$$\overline{\sigma} = \sigma.$$

Equivalently, we can define the closure of a simplex $\sigma \in \mathsf{K}$ (simplex set $\Sigma \subset \mathsf{K}$) as the smallest subcomplex of K containing σ (including Σ). The *link* of a simplex σ (see Figure 2.2, bottom) is defined as the set of all the the simplices in the closure of the star of σ, which do not have σ as a face:

$$\mathrm{lk}(\sigma) = \mathrm{cl}(\mathrm{st}(\sigma)) - \mathrm{st}(\mathrm{cl}(\sigma)).$$

It can be proven that for every simplex $\sigma \in \mathsf{K}$, $\mathrm{lk}(\sigma)$ is a subcomplex.

The *carrier* $\|\mathsf{K}\|$ of a simplicial complex K (also called the *polyhedron* $\|\mathsf{K}\|$) is a subset of E^n defined by the union, as point sets, of all the simplices in K. For each point $v \in \|\mathsf{K}\|$ there exists exactly one simplex $\sigma \in \mathsf{K}$ containing v in its relative interior. This simplex is denoted by $\mathrm{supp}(v)$ and called the *support* of the point v.

We say that a point $\mathbf{v} \in A \subset E^n$ is *p-manifold* if there exists a neighbourhood U of v such that $A \cap U$ is homeomorphic to $\mathbb{R}^p$ or $\mathbb{R}^{(p-1)} \times (0, +\infty)$. Otherwise we call $\mathbf{v}$ *non-manifold*. We say that a simplex $\sigma \in \mathsf{K}$ is p-manifold, if every point of the relative interior of this simplex is p-manifold with regard to the carrier of K. E.g. obviously each n-simplex is n-manifold, each $(n-1)$-simplex is n-manifold if it is a face of at least one n-simplex, etc. We also say that an n-dimensional simplicial complex K in E^n is manifold, if each of its simplices is n-manifold.

We say that a $(n-1)$-simplex $\sigma^{n-1} \in \mathsf{K}$ is *boundary* in n-dimensional simplicial complex K if it is not n-manifold or is a face of only one n-simplex in K. We say that a p-simplex $\sigma^p \in \mathsf{K}$ $(p < n-1)$ is boundary in K if it is not n-manifold or is a face of at least one boundary $(p+1)$-simplex in K. The union of all boundary simplices in K forms the *topological boundary* of K according to the Euclidean topology in E^n.

2.1.2 Abstract simplicial complexes

Alongside Euclidean simplicial complexes, we define *abstract simplicial complexes*, which provide a purely combinatorial description of Euclidean simplicial complexes, regardless of their geometry. In other words, abstract simplicial complexes describe the *connectivity* of Euclidean simplicial complexes. They are important from a software engineer's point of view, because while designing a data structure to represent simplicial complexes, we often want to decouple geometric data from the connectivity information. Many of the notions used to describe Euclidean simplicial complexes generalize easily to abstract simplicial complexes.

An *abstract simplicial complex* (compare [57, p. 96-101]) $\mathcal{K}$ is a collection of nonempty finite *sets* (called *abstract simplices*), such that:

$$\sigma \in \mathcal{K} \ \wedge \ \emptyset \neq \sigma' \subset \sigma \ \Rightarrow \ \sigma' \in \mathcal{K}.$$

A set $\mathcal{K}' \subset \mathcal{K}$, for which the condition above also holds, is called a *subcomplex* of $\mathcal{K}$. We call the set $\mathcal{V} = \bigcup_{\sigma \in \mathcal{K}} \sigma$ a set of *vertices* of $\mathcal{K}$, any element of a simplex $\sigma \in \mathcal{K}$ a *vertex* of σ, and any subset of a simplex $\sigma \in \mathcal{K}$ a *face* of σ (we assume equivalence between a vertex v and the face $\{v\}$). For $\sigma \in \mathcal{K}$ we define:

$$\dim(\sigma) = |\sigma| - 1.$$

The dimension of $\mathcal{K}$ is the maximum dimension of any simplex in $\mathcal{K}$, if it exists, otherwise we say that $\mathcal{K}$ is *infinite-dimensional*. We say that $\mathcal{K}$ is a *finite complex* if $\mathcal{K}$ is a finite set. We can easily generalize the notions of *closure*, *star* and *link* for the abstract simplicial complexes. The closure of a simplex $\sigma \in \mathcal{K}$ is the smallest subcomplex of $\mathcal{K}$ that contains σ, that means:

$$\mathrm{cl}(\sigma) = \{\sigma' \in \mathcal{K} : \sigma' \subset \sigma\}.$$

The *star* of $\sigma \in \mathcal{K}$ is the set of all simplices that have σ as a face:

$$\mathrm{st}(\sigma) = \{\sigma' \in \mathcal{K} : \sigma' \supset \sigma\}.$$

The *link* of $\sigma \in \mathcal{K}$ can be defined as follows:

$$\mathrm{lk}(\sigma) = \mathrm{cl}(\mathrm{st}(\sigma)) - \mathrm{st}(\mathrm{cl}(\sigma)).$$

Having a finite Euclidean simplicial complex K we can easily construct an abstract simplicial complex $\mathcal{K}$ by identifying each simplex $\sigma \in \mathsf{K}$ with the set of its vertices.

2.1.3 Orientations and orientability

We introduce the following equivalence relation in the set P_σ of all orderings $(\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_{p+1}})$ of the vertices of $\sigma = \langle \mathbf{v}_1, \ldots, \mathbf{v}_{p+1} \rangle$:

$$(\mathbf{v}_{i_1}, \mathbf{v}_{i_2}, \ldots, \mathbf{v}_{i_p}) \sim (\mathbf{v}_{\pi(i_1)}, \mathbf{v}_{\pi(i_2)}, \ldots, \mathbf{v}_{\pi(i_p)})$$
$$\Updownarrow$$
$$\pi : \{1, 2, \ldots, p+1\} \to \{1, 2, \ldots, p+1\} \text{ is an even permutation operator.}$$

We call each element of the quotient set $\mathcal{O}_\sigma = P_\sigma/\!\sim$ (that means each equivalence class in P_σ with regard to equivalence relation $\sim$) an *orientation* of a simplex σ. It is easy to notice, that if $p > 0$, $|\mathcal{O}_\sigma| = 2$, which means that there are only two possible orientations for any simplex defined on a set of $p+1$ points from E^n. If $p = 0$, obviously $|\mathcal{O}_\sigma| = 1$.

An *oriented simplex* is a simplex together with a choice of orientation. We will write $[\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}]$ for the p-simplex oriented by the vertex ordering $(\mathbf{v}_1, \ldots, \mathbf{v}_{p+1})$, and we will let $-[\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}]$ denote the same simplex with the opposite orientation. Thus, for example, for 2-simplices we have

$$[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] = [\mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_1] = [\mathbf{v}_3, \mathbf{v}_1, \mathbf{v}_2] = -[\mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_2] = -[\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1] = -[\mathbf{v}_2, \mathbf{v}_1, \mathbf{v}_3].$$

The orderings belonging to the same equivalence class of $\mathcal{O}_\sigma$ are simply those orderings, for which the sign of the volume of the simplex is the same. We define an *oriented volume* of $\sigma = [\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}]$

$$\mathcal{V}(\sigma) = \mathcal{V}(\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}) = \frac{1}{p!} \det(\mathbf{v}_1 - \mathbf{v}_2, \mathbf{v}_2 - \mathbf{v}_3, \ldots, \mathbf{v}_p - \mathbf{v}_{p+1}, \mathbf{v}_{p+1} - \mathbf{v}_1).$$

It can be proven, that for an even permutation operator π

$$\mathcal{V}(\mathbf{v}_{\pi(1)}, \ldots, \mathbf{v}_{\pi(p+1)}) = \mathcal{V}(\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}),$$

and for an odd permutation operator π'

$$\mathcal{V}(\mathbf{v}_{\pi'(1)}, \ldots, \mathbf{v}_{\pi'(p+1)}) = -\mathcal{V}(\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}),$$

hence we can see that the combinatorial definition of the orientation corresponds naturally to the geometric definition (according to which the orientation is determined by the sign of the oriented volume of a simplex). The orientation of σ, for which $\mathcal{V}(\sigma) > 0$ is called the *natural orientation*.

The p-simplex $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{p+1}]$ determines an orientation of each of its $(p-1)$-faces, called the *induced orientation*, by the following rule: the induced orientation on the face $\sigma_i^{p-1} = \langle \mathbf{v}_1, \ldots, \mathbf{v}_{i-1}, \mathbf{v}_{i+1}, \ldots, \mathbf{v}_{p+1} \rangle$ is defined to be $(-1)^{i+1}[\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}, \mathbf{v}_{i+1}, \ldots, \mathbf{v}_{p+1}]$. Now suppose K is an n-dimensional simplicial complex in which every $(n-1)$-simplex is a face of no more than two n-simplices. If σ_i^n and σ_j^n are two n-simplices that share an $(n-1)$-face σ^{n-1}, we say that orientations of σ_i^n and σ_j^n are *consistent* if they induce opposite orientations on σ^{n-1}. An orientation of K is a choice of orientation of each n-simplex in such a way that any two simplices that interesect in an $(n-1)$-face are consistently oriented. If a complex K admits an orientation, it is said to be *orientable*.

2.1.4 Triangle Meshes

A dimension 2 simplicial complex $\mathsf{K} \subset E^n$ (where $n \geq 2$), such that every 0 or 1-simplex $\sigma \in \mathsf{K}$ is a face of a 2-simplex $\sigma^2 \in \mathsf{K}$ is called a *triangle mesh*. Triangle meshes inherit the notions of manifoldness and orientability from simplicial complexes. Let us concentrate on orientable triangle meshes. We recognize the following useful, local operations on triangle meshes (illustriated in Figure 2.3):

- *edge flip* (also known as *edge swap*) – replaces an edge e (and two adjacent triangles) with an edge connecting the vertices in the link of e (and introduces two new triangles);
- *edge split* – one of the tools for mesh *refinement*, introduces a new vertex on the edge which splits the edge and two adjacent triangles;
- *triangle split* – another tool for mesh *refinement*, introduces a new vertex inside a triangle which splits it into 3 new triangles;
- *edge collapse* – a tool for mesh *simplification*, removes the edge e and two adjacent triangles by identifying the vertices of e;

Edge collapse and edge flip have to be used with caution, as they are prone to destroy the simplicial complex property. Edge collapse might also destroy the manifoldness of the mesh (the criteria for when edge collapse can be used safely have been presented in [71]).

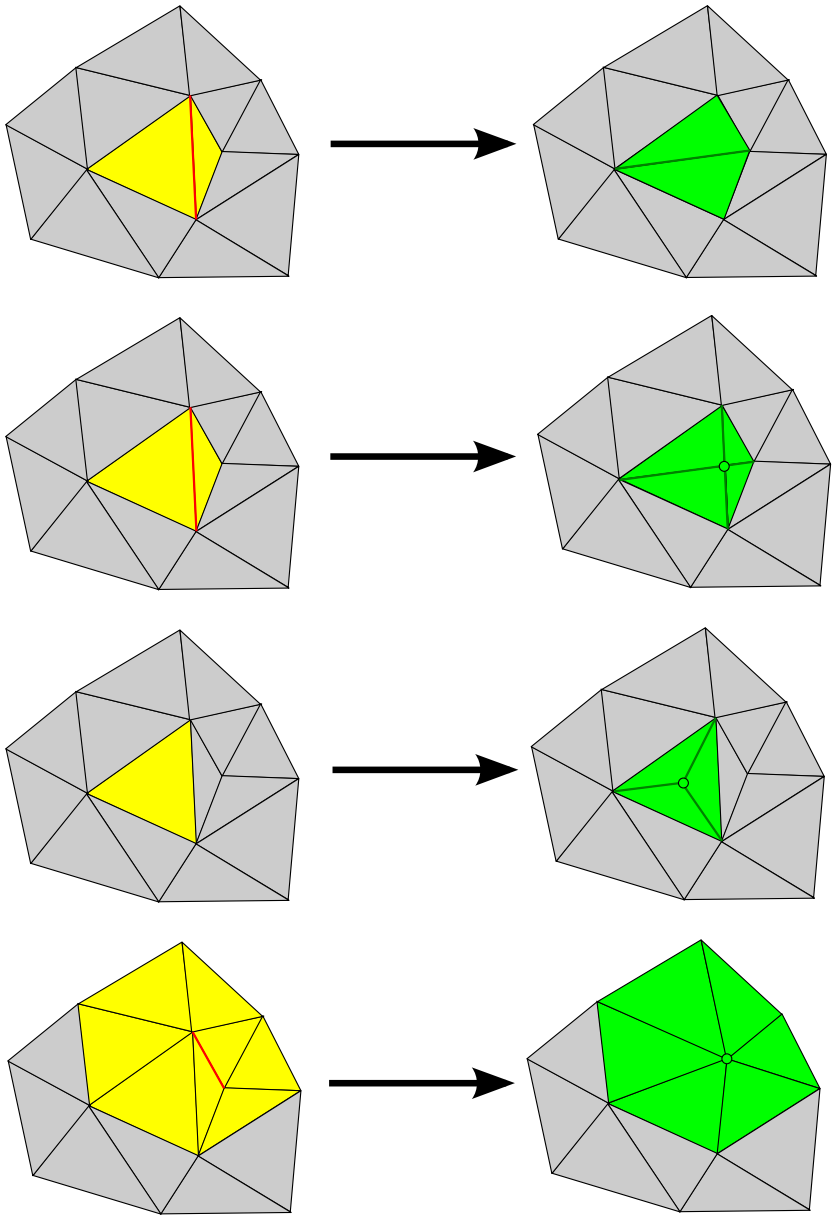

Figure 2.3: Operations on triangle meshes, from top to bottom: edge flip, edge split, triangle split and edge collapse.

2.1.5 Tetrahedral Meshes

A dimension 3 simplicial complex $\mathsf{K} \subset E^n$ (where $n \geq 3$), such that every 0, 1 or 2-simplex $\sigma \in \mathsf{K}$ is a face of a 3-simplex $\sigma^3 \in \mathsf{K}$ is called a *tetrahedral mesh*. Tetrahedral meshes inherit the notions of manifoldness and orientability from simplicial complexes. Again, we concentrate our attention only on orientable tetrahedral meshes. The operations on tetrahedral meshes generalize aforementioned operations on triangle meshes, e.g.: edge split, face (triangle) split, tetrahedron split, edge collapse. However an edge flip (reconnection) can be generalized in several ways. Those and more details on the subject of tetrahedral meshes are presented in Chapter 3.

2.2 Implementation

Our implementations of both 2D and 3D deformable simplicial Complexes decouple the topological level (connectivity, boundary and co-boundary relations, traversal, reconnection – *abstract simplicial complex*) from the geometric level (spatial positions of vertices, validity check, geometric algorithms). In the 2D case we are using the popular *half-edge* data structure [20, 39, 62] (used in e.g.: CGAL [9] and OpenMesh [10]) to represent triangular meshes (as implemented in GEL library [5]). Half-edge data structure provides simple mesh traversal, and allows the performance of topological operations easily. It is, however, not suitable for representing non-manifold meshes.

The implementation of 3D deformable simplicial complexes was designed in a way that allows representing non-manifold simplicial complexes and gives a possibility to undo the changes done to the mesh. We were initially planning to use the data structure also for other tetrahedral mesh based applications, so we did not want restrict ourselves to manifold meshes. We were also considering using *try and roll-back* approach to mesh quality improvement (first try performing an operation on the mesh and if it improves the quality locally—accept it, if it does not—undo), for example by using *simulated annealing* [49].

We store the traits data for each of the simplices in one of four array-based kernels: one for vertices, edges, faces and tetrahedra (see Figure 2.4). Simplicial complex connectivity is recorded in the *incidence simplicial* data structure [30], which supports efficient traversal and allows to perform topological operations quite easily.

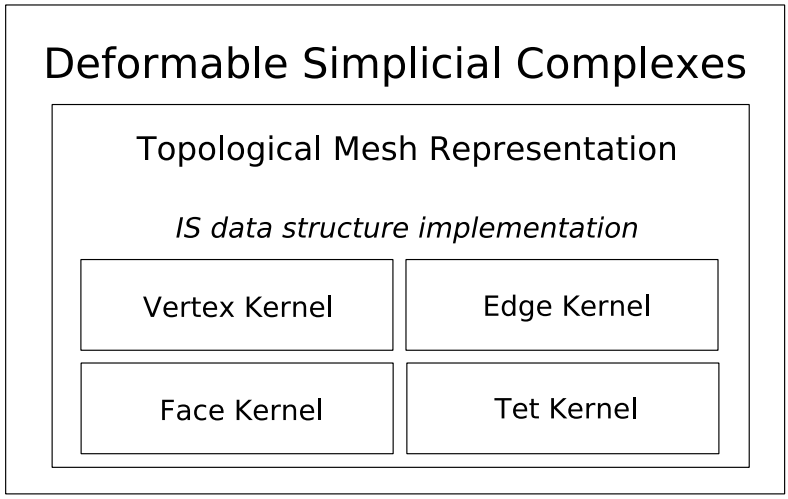

Figure 2.4: Main components of the 3D deformable simplicial complexes implementation.

2.2.1 Kernel

The *kernel* is a low-level data structure which is responsible for handling memory allocation, making it transparent to the user. The very basic operations the kernel should support are creating a new object and deleting an object.

Our generic kernel is array based. Arrays elements (*cells*) are identified by *keys*, which is simply their indices. Each array element consists of:

- type traits – data depending on what kind of object is stored in the kernel;
- label stating whether the element is *used*, *marked* for deletion or *empty*;
- indices to the previous, and the next element with the same label – meaning that there are doubly-linked lists defined over the array.

Whenever a new object is **created**, it is stored in an array element taken off the front of the *empty* list and added at the end of the *used* list. If the *used* list becomes empty, the array size is doubled. Whenever an object is **removed**, it is taken out of the *used* list and added at the end of the *marked* list. The *marked* list is cleared when we **commit** the changes. All allocated data stored in *marked* cells gets deallocated, all *marked* cells are added to the *empty* list, and the lists are reconnected so that they follow the ordering given by the indices.

Undo functionality. Our kernel implementation provides light-weight, multiple-level undo functionality. Whenever we plan an operation that we might want to undo, we have to identify the set of objects that might become changed in the process. We call the kernel to **set an undo mark**, which consists of a copy of the tail of the *used* list, a copy of the tail of the *marked* list and a list of copies of cells containing selected objects. An undo mark is added on the top of the *undo stack*. Whenever **undo** command is called, an undo mark is taken off the top of the undo stack, the lists are reconnected (reconnection scheme utilises the fact that we always add new elements at the end of *used* and *marked* lists) and saved trait data is restored. We can also call **undo all** command, which performs the undo operation as long as the undo stack is not empty. **Commit** command, besides the functionality described above, clears the undo stack.

Kernel implementation was delegated to a master student Frederik Gottlieb, Department of Computer Science, Univeristy of Copenhagen [37].

2.2.2 Incidence Simplicial Data Structure

The *incidence simplicial* (IS) data structure [30] is a dimension-independent, compact data structure designed for representing arbitrary simplicial complexes. However, our implementation restricts to the 3D case, so from this point we assume that $n = 3$. In the IS data structure, with each p-simplex σ^p (for $p > 1$) the unordered set of handles to its $p{+}1$ $(p{-}1)$-dimensional faces $\sigma_1^{p-1}, \ldots, \sigma_{p+1}^{p-1}$ is stored (the boundary relation $B_{p,p-1}(\sigma^p)$). However, since our implementation is orientation-aware, we identify an oriented simplex σ^p with an *ordered* tuple of its $(p-1)$-faces:

$$\left[\sigma_1^{p-1}, \ldots, \sigma_{p+1}^{p-1}\right],$$

which implies:

$$\sigma^p = \left[\mathrm{vert}\big(\sigma^p\big) - \mathrm{vert}\big(\sigma_1^{p-1}\big), \ldots, \mathrm{vert}\big(\sigma^p\big) - \mathrm{vert}\big(\sigma_{p+1}^{p-1}\big)\right],$$

where:

$$\mathrm{vert}\big(\sigma^d\big) = \bigcup_{i=1}^{d+1} \mathrm{vert}\big(\sigma_i^{d-1}\big),$$

and we identify $\{v\}$ with $\langle v\rangle$, $[v]$ and v, where v is a vertex handle. In order to make the traversal efficient, *partial coboundary relation* $C^*_{p,p+1}(\sigma^p)$ is also stored with every p-simplex σ^p, for $p < n$. Partial coboundary relation $C^*_{p,p+1}(\sigma^p)$ consists of $(p+1)$-simplices from $\mathrm{st}(\sigma^p)$ connecting σ^p with its link, one per each connected component in $\mathrm{lk}(\sigma^p)$. Hence, it can be easily seen that:

- $C^*_{2,3}(\sigma^2) = C_{2,3}(\sigma^2)$,
- if σ^p $(p < 2)$ is 3-manifold, then $|C^*_{p,p+1}(\sigma^p)| = 1$.

Unlike the authors of the original structure, we assume that either full or partial coboundary relation can be stored with the simplex σ, since some of the topological operations require evaluating the full coboundary relations of certain simplices, and we would prefer to avoid *uncompressing* (evaluating the full coboundary relation) and *compressing* (evaluating the partial coboundary relation based on the full coboundary relation) the coboundary relation of a simplex everytime we perform those topological operations on it. Hence we also need an one-bit flag indicating whether full coboundary relation (*uncompressed* state) or partial coboundary relation (*compressed* state) is stored. We define the *compression rate* of a simplicial complex K represented with the IS data structure as a ratio:

$$\frac{\|\{\sigma \in \mathsf{K} : \sigma \text{ in compressed state}\}\|}{\|\mathsf{K}\|}.$$

Whenever the compression rate drops below some desired value given by the user, the simplices in an uncompressed state have their partial coboundary evaluated and stored.

We have implemented several operations for traversal and manipulation of the simplicial complex, including:

- **star** – evaluation of the star of a simplex;
- **closure** – evaluation of the closure of a simplex or a set of simplices;
- **link** – evaluation of the link of a simplex;
- **boundary** – evaluation of the boundary of the simplex;
- **orient faces consistently/oppositely** – enforcing a consistent/opposite orientation on all $(p-1)$-faces of a p-simplex σ^p;
- **orient co-faces consistently/oppositely** – enforcing a consistent/opposite orientation on all $(p+1)$-simplices having a given p-simplex σ^p as a face;

Besides those we have implemented all topological operations described in Chapter 3.

2.2.3 Discussion

Looking back, we have came to conclusion that a few mistakes in the planning phase were made. First of all, we are only using manifold tetrahedral meshes in the deformable simplicial complexes. We could have utilised this fact in simplifying the IS data structure, since in manifold tetrahedral meshes the partial coboundary relation is fixed size (one element) for vertices and edges, and has either one or two elements for faces. Moreover, we could have employ manifoldness in designing more efficient (and less error-prone) traversal algorithms than the ones proposed by Hui and de Floriani. Furthermore, it turned out that in most of the cases, using the *try and roll-back* approach to some mesh operations is not time efficient, and we ended up not using the undo functionality of the kernel. We believe that by designing a kernel without undo functionality we could have saved quite a lot of time, as this was the most error-prone part of the implementation. Finally, it turned out that the task of implementing the kernel exceeded the time frame our student was given to complete his thesis and we ended up having to test and debug his code ourselves. Nevertheless, we believe that our perfectly generic kernel could be used in other applications, where the demand for the roll-back possibility is more evident.

Chapter 3

Tetrahedral Mesh Improvement Using Multi-face Retriangulation

Marek Krzysztof Misztal, Technical University of Denmark
Jakob Andreas Bærentzen, Technical University of Denmark
François Anton, Technical University of Denmark
Kenny Erleben, University of Copenhagen

Published in *Proceedings of the 18th International Meshing Roundtable*, Salt Lake City 2009.

Abstract. In this paper we propose a simple technique for tetrahedral mesh improvement without inserting Steiner vertices, concentrating mainly on boundary conforming meshes. The algorithm makes local changes to the mesh to remove tetrahedra which are poor according to some quality criterion. While the algorithm is completely general with regard to quality criterion, we target improvement of the dihedral angle. The central idea in our algorithm is the introduction of a new local operation called multi-face retriangulation (MFRT) which supplements other known local operations. Like in many previous papers on tetrahedral mesh improvement, our algorithm makes local changes to the mesh to reduce an energy measure which reflects the quality criterion. The addition of our new local operation allows us to advance the mesh to a lower energy state in cases where no other local change would lead to a reduction. We also make use of the edge collapse operation in order to reduce the size of the mesh while improving its quality. With these operations, we demonstrate that it is possible to obtain a significantly greater improvement to the worst dihedral angles than using the operations from the previous works, while keeping the mesh complexity as low as possible.

3.1 Introduction and Motivation

For many types of physical simulation, the tetrahedral mesh representation is the natural choice. For instance, finite element computations in 3D usually employ tetrahedral meshes which are far better at adapting to boundaries and changes in scale than e.g. regular voxel grids.

For 2D triangulations, Delaunay triangulation is often a natural choice since it leads to a mesh which is optimal in the sense that the minimal angles are maximized which is a reasonable quality criterion in 2D. In 3D however, it is less clear what quality criterion we should strive for and a 3D Delaunay tetrahedralization may contain very flat sliver tetrahedra with extreme dihedral angles, and extreme dihedral angles are often precisely what we wish to avoid since they may lead to problems, such as great interpolation errors or ill-conditioned stiffness matrices in some finite element computations (although in the anisotropic case they might be desirable) or problems with interpolation accuracy [88].

Consequently, in this paper and in other recent work [52], the goal is to optimize a tetrahedralization obtained through either Delaunay or other methods in order to improve some criterion – particularly dihedral angles. However, little is known about globally optimal meshes in the sense that the smallest dihedral angle is maximal or that the largest dihedral angle is minimal. Consequently, one strives

instead for a set of simple, local transformations which improve the mesh by removing poorly shaped tetrahedra. The best one can hope for in this case is a good local minimum, and whether one attains such a minimum is highly dependent on one's vocabulary of local transformations. It is this vocabulary which we extend by the addition of two local transformations which are highly beneficial to the mesh quality yet have not previously been used in tetrahedral mesh optimization.

The most powerful way of improving triangle or tetrahedral meshes is through the insertion of more vertices (as shown in [52]). Indeed this is sometimes the only way to improve quality. Unfortunately, one pays the price of adding (sometimes significantly) more tetrahedra, and finding the optimal place to put a vertex can be hard. Besides, many applications (such as dynamic meshes) require their own Steiner vertex insertion routines. For these reasons, we opine that it is very worthwhile to explore to what extent our mesh improvement vocabulary can be augmented without adding vertices.

Our main contribution is the *multi-face retriangulation* operation. Assume a set of tetrahedra which we can divide into *upper* and *lower* tetrahedra. Any upper tetrahedron shares a face with precisely one lower tetrahedron (and vice versa) and the upper tetrahedra all share a vertex (the upper vertex) as do the lower tetrahedra (the lower vertex). We can say that the set of tetrahedra is *sandwiched* between the upper and the lower vertex (as illustrated in Figure 3.1). The union of the triangular faces shared between upper and lower tetrahedra can be seen as a triangulation of a polygon. Our proposed operation simply retriangulates this polygon to obtain better sets of upper and lower tetrahedra. Multi-face retriangulation can also be seen as a composition of the known *multi-face removal* and *edge removal* operations (as shown in Figure 3.1) [34, 52]. However, multi-face retriangulation is more powerful than the concatenation of these two operations: in the case of some configurations multi-face removal followed by edge removal would never be selected because very poor or inverted tetrahedra would result from the multi-face removal operation (as illustrated in Figure 3.2). Additionally, multi-face retriangulation works on boundaries whereas concatenation of multi-face removal and edge removal does not.

The other contribution is the use of the well known *edge collapse* operation. Curiously, to the best of our knowledge, this operation has not been incorporated into any tetramesh improvement algorithm previously. It significantly reduces the complexity of the mesh and it also might improve the worst quality within the set of affected tetrahedra.

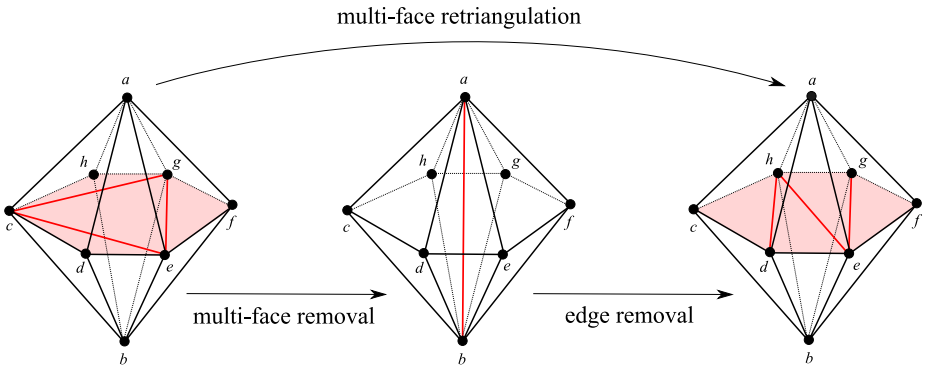

Figure 3.1: Multi-face removal, edge removal and their superposition – multi-face retriangulation.

3.2 Related Work

Clearly, whether mesh improvement is needed depends on how the mesh was generated. Broadly speaking, there are three ways of producing tetrahedral meshes from a boundary representaion of an object. First, if the boundary is a piecewise linear complex (in particular – triangulated manifold), we could use constrained Delaunay tetrahedralization to produce a conforming mesh [86, 89]. Alternatively, we could use an advancing fronts method which would build the tetrahedralization out from the boundary. As mentioned, the former approach will often have problems with sliver tetrahedra even after Delaunay refinement, unless the boundary satisfies a set of strict conditions [18, 86] limiting the practical applications of this approach, and the latter tends to produce some bad tetrahedra around areas where the front collides on itself [70]. These problems are compounded if the boundary mesh has poorly shaped triangles. An alternative approach is the centroidal Voronoi tessellation based Delaunay tetrahedralization [22] which can, however, still leave some poorly shaped tetrahedra. We conclude that the mesh optimization is likely to be useful as a step following both Delaunay based methods and also advancing fronts based methods.

A third and alternative strategy is to force the boundary to conform to an isosurface of an implicit function rather than a mesh. The spatial domain is first divided into tetrahedra, and a subset which approximates the shape well is selected. In a subsequent compression step, the boundary vertices of this subset are forced to lie precisely on the isosurface [69]. However, we note that the compression step is an optimization procedure because, generally, not only the boundary vertices are moved but also the interior vertices in order to improve the

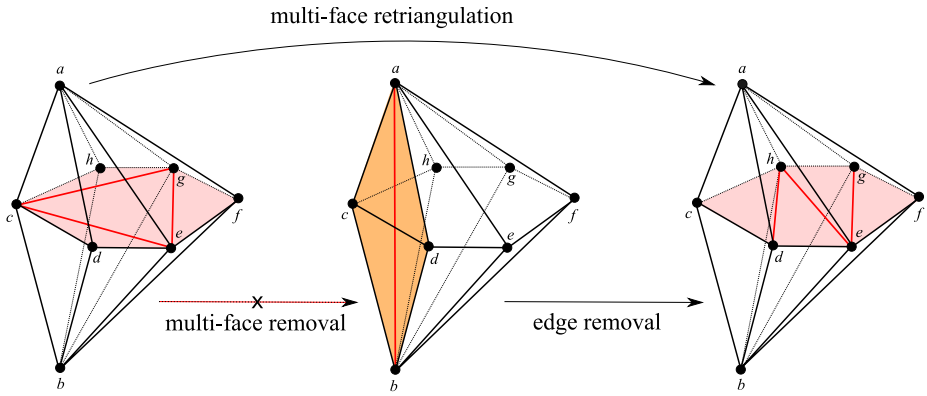

Figure 3.2: Configuration in which multi-face removal would not be performed. Vertices a, b, c and d are nearly coplanar. Performing multi-face removal in such a configuration would lead to creating a very poorly shaped tetrahedron $abcd$ (highlighted in orange) of extremely low quality. Also, by perturbing vertex a or b we can easily create a situation in which tetrahedron $abcd$ would be inverted. In both situations multi-face removal would not be performed by a greedy algorithm – hence the arrow is crossed out. The strength of multi-face retriangulation is in tunneling through these kinds of hills in the energy landscape.

quality of the mesh. In recent work, Labelle and Shewchuk were able to demonstrate good provable bounds on the dihedral angles using such a method [54]. However, methods which fit meshes to isosurfaces [54, 69] cannot be expected to capture sharp edges and corners because the vertices are not constrained to lie in particular positions. Consequently, in some cases they simply do not apply.

Most of the existing work for tetrahedral mesh improvement uses the following three types of mesh operations:

1. *Mesh smoothing* – relocation of the mesh points in order to improve mesh quality without changing mesh topology.

2. *Topological operations* – reconnection of the vertices in the mesh (without displacing them).

3. *Vertex insertion* – adding extra vertices into the mesh (through eg. splitting of the edges, faces or tetrahedra) and reconnecting affected regions of the mesh.

3.2.1 Mesh Smoothing

One of the best known smoothing methods, Laplacian smoothing, in which a vertex is moved to the centroid of the vertices to which it is connected, is a popular and quite effective choice for triangular meshes. In tetrahedral meshes, however, it often produces poor tetrahedra [32]. Optimal (with regard to linear interpolation error) Delaunay vertex placement has been investigated by Chen and Xu [15]. More general mesh smoothing algorithms are based on numerical optimization. One of the most popular local algorithms for mesh smoothing was suggested by Freitag et al. [33]. This method relocates one vertex at a time. Given one vertex, its new position is found, so that the minimum quality of all the tetrahedra adjacent to this vertex is maximized (this requires non-smooth optimization). This procedure is performed for each vertex in the mesh and can be iterated until a stable configuration is attained. It can also be performed on the boundary of the mesh, given extra constraints for the position of the vertex. Another optimization based approach, using generalized linear programming, was presented by Amenta et al. [1], but this one is not as general as Freitag's and is not well suited for dihedral angles optimization. Mesh smoothing can also be performed by continuous optimization in the space of coordinates of all vertices of the mesh (as in [40]), but Freitag's method has advantages over this approach – it is easier to use with non-smooth quality measures, and its characterised by stable behavior even if the initial quality of the mesh is very low.

3.2.2 Topological Operations

Reconnection of the mesh can be pictured as picking a set of adjacent tetrahedra and replacing them with another set of tetrahedra, of higher minimum quality, filling in the same volume. This can be performed in a more or less arbitrary manner (as small polyhedron re-tetrahedralization in [59]), or can be organized into a set of *topological operations*, such as:

- *2-3 flip* and its inverse, *3-2 flip*, as shown in Figure 3.3.
- *4-4 flip* and its version for boundary configuration, *2-2 flip*, illustrated in in Figure 3.3 – ambiguous, requires specifying which edge pair of vertices is going to be connected after the operation.
- *Edge removal* is illustrated in Figure 3.1 – generalizes 3-2 flip, 4-4 flip and 2-2 flip; ambiguous, requires specifying the final triangulation of the link of the removed edge, which can be performed by using triangulation

templates, as in [34] or by using Klincsek's algorithm [50] in order to maximize the minimal quality of the created set of tetrahedra, as in [52]. Edge removal can be performed for boundary edges.

- *Multi-face removal* of de Cougny and Shephard [21] is the inverse to the edge removal, as shown Figure 3.1 – generalizes 2-3 flip and 4-4 flip; requires dynamic programming in order to select the subset of faces sandwiched between two vertices, which gives the best improvement.

The original paper of Freitag and Ollivier-Gooch [34] uses the first three operations. It can easily be noticed, that multi-face removal can actually be decomposed into a sequence of a single 2-3 flip followed by a certain number of 3-2 flips. However, this can not always be performed in the *hill-climbing* approach (which is usually the choice for the tetrahedral mesh-improvement algorithms) if one of the operations in the sequence decreases quality locally. Klingner and Shewchuk [52] use all the operations from the list above.

3.2.3 Vertex Insertion

Klingner and Shewchuk showed in [52] that mesh improvement is far more effective with the inclusion of transformations that introduce Steiner vertices. Proper placement of Steiner vertices is a hard problem. Klingner and Shewchuk describe a sophisticated and rather complex algorithm for vertex insertion which mimics Delaunay vertex insertion and, together with optimization based smoothing and topological operations, allows them to improve the meshes so that all dihedral angles are between 31° and 149°, or, using a different objective function, between 23° and 136°.

3.3 Tetrahedral Mesh Quality Improvement

Our mesh improvement algorithm is based on the algorithm proposed by Klingner and Shewchuk [52] (which in turn extends one by Freitag and Ollivier-Gooch [34]) which uses vertex smoothing by Freitag et al. [33], edge removal, multi-face removal and vertex insertion (most of the operations they use can be performed on the boundary of the mesh). In turn, our algorithm uses the following set of operations:

- Vertex-smoothing as in Freitag et. al [33],

- Topological operations:
  - edge removal,
  - multi-face removal,
  - multi-face retriangulation.
- Edge collapse.

Vertex smoothing and edge removal can be performed for the boundary vertices and edges, if the boundary is sufficiently flat around them. Additionally, vertex smoothing can be performed along *straight* ridges on the boundary of the mesh, if the surface patches separated by the ridge are sufficiently flat. Multi-face removal and edge removal are implemented essentially the same way as in [87].

3.3.1 Multi-face Retriangulation

Multi-face retriangulation can be seen as a composition of multi-face removal and edge removal, however, it can be also performed on the boundary of the mesh. It includes the 4-4 and 2-2 flips. Multi-face retriangulation does not change the number of tetrahedra in the mesh. So far as we know, it has never appeared in the literature.

The reasons in favor of using MFRT alongside multi-face removal and edge removal are:

- In some cases, the configuration produced by multi-face removal is of lower quality, as illustrated in Figure 3.2. Thus a greedy approach would not select that configuration even if the subsequent edge removal led to a state of lower energy than the initial configuration.
- In some cases the configuration produced by multi-face removal includes inverted tetrahedra, and no approach would select that (also shown in Figure 3.2). However, MFRT cannot produce inverted tetrahedra, as the best triangulation of the multi-face cannot be worse than the initial one and we assume we run our algorithm on valid tetrahedral meshes.
- MFRT can be applied to boundary configurations. To see this, let us only consider a set of lower tetrahedra in Figure 3.1. In such a configuration, multi-face consists of boundary faces and it cannot be removed using multi-face removal, but it can easily be retriangulated. However, if the multi-face is not sufficiently flat, which is the case when the angles between the

normals to the faces are greater than 0°, MFRT can change the geometry of the boundary of the mesh, which is usually not desirable.

- MFRT does not change the number of tetrahedra. This property is a direct consequence of a well known fact that every triangulation of a polygon has the same number of triangles.

In our implementation, the input is a single face f we wish to remove. We find the apices a and b of the two tetrahedra adjoining f. Among the set of all faces sandwiched between a and b we find the connected component that contains f. For the multi-face defined like that, we find the optimal triangulation of its bounding polygon using Klincsek's algorithm. The routine is similar for a boundary face f, although in this case we have to make sure that the retriangulated multi-face is sufficiently flat (otherwise geometry of the boundary might change).

3.3.2 Edge Collapse

Edge collapse (also known as *edge contraction* or *half-edge contraction*) is a well known mesh operation which has been used as a primary tool for simplifying 2D and 3D meshes in numerous works, such as [19, 71]. It identifies one of the vertices of an edge e with the other, removes e and all faces and tetrahedra which contain it. This can, however, lead to invalid configurations (violating the simplicial complex criterion) or alter the surface geometry of the mesh, unless certain conditions are fulfilled, described in detail by Natarajan and Edelsbrunner in [71]. If edge collapse is not performed for the boundary edges, which is the case in our implementation, those conditions simplify to the following:

$$\mathrm{Lk}(e) = \mathrm{Lk}(a) \cap \mathrm{Lk}(b),$$

where a and b are the vertices of the edge e, and $\mathrm{Lk}(\sigma)$ denotes the link of a simplex σ which, in tetrahedral meshes, can be defined as a set of those simplices (vertices, edges and faces) in the mesh, that do not intersect with σ, but are contained by the one of the tetrahedra containing σ. In our implementation this is performed if the minimum quality of the set of tetrahedra affected by the operation increases, or if it does not decrease below a certain quality threshold q_{min}, which is a global parameter of our algorithm.

3.3.3 Quality Measures

Both the smoothing algorithm and the topological operations which we are using are indifferent to the tetrahedron quality measure. In order to be able to compare our results to those provided in [52] and [34], we are using:

- The *minimum sine measure* – the minimum sine of a tetrahedron's six dihedral angles, penalizes both small and large dihedral angles.
- The *minimum biased sine measure*, which is like the minimum sine measure, but if a dihedral angle is obtuse, its sine is multiplied by 0.7 (before choosing the minimum). This quality measure penalizes large angles more aggressively than the small angles.

Many quality measures have been proposed for tetrahedral meshes reviewed by [88], [40]. Our two choices are well behaved and very intuitive, although non-smooth.

3.4 Implementation

Our mesh improvement schedule follows that of Klingner and Shewchuk [52] (pseudo code is shown in Algorithm 1). Same as in their work, we use a short list of *quality indicators* in order to measure progress in lowest quality tetrahedra improvement. Those are: the quality of the worst tetrahedron in the entire mesh and seven *thresholded means* of the qualities of all the tetrahedra in the mesh. A mean $\bar{q_\theta}$ with threshold θ is computed the following way:

$$\bar{q_\theta} = \frac{1}{\#\{\text{tetrahedra in } M\}} \sum_{t \in M} \min(q(t), \theta),$$

where M is the mesh and q is the tetrahedron quality measure we use. For our quality measures we use thresholded means with thresholds $\sin(1°)$, $\sin(5°)$, $\sin(10°)$, $\sin(15°)$, $\sin(25°)$, $\sin(35°)$ and $\sin(45°)$. A quality indicator designed like that is a good measure of how narrow the distribution of the tetrahedron qualities is and allows us to detect the quality improvement even if the minimum quality does not change. The minimum quality alone is much less efficient as a mesh quality indicator – it leads to premature termination of the mesh improvement algorithm and significantly worse final results. We consider the improvement in the mesh quality *sufficient* if either the quality of the worst tetrahedron improves, or if one of the thresholded means increases by at least 0.0001.

We begin mesh improvement with a vertex smoothing pass, followed by a topological pass. In the topological pass, pseudo code of which is shown in Algorithm 2, we first obtain the list of all the tetrahedra in the mesh and then try to remove every tetrahedron t on the list by first trying to remove its edges using the edge remove operation and then, if we have not succeeded, by trying to remove its faces using multi-face retriangulation followed by multi-face removal. Such an ordering of the operations is justified by the fact that first performing multi-face retriangulation still leaves room for extra improvement through multi-face removal, while it does not work the other way round. The optimal multi-face for multi-face removal is chose using dynamic programming, accordingly to an algorithm described in [87]. Any of those operations are performed only if they locally improve the quality. If that happens, we proceed to the next tetrahedron on the list. Every topological operation that we use can destroy more tetrahedra than the one for which it was called, so before attempting to remove any tetrahedron we have to make sure that it still exists in the mesh.

After two initial passes we begin the main loop, in which we first smooth all the vertices until there is no more improvement detected by our mesh quality indicators. Then we start the topological pass again. If it improves the quality of the mesh sufficiently, we start the loop again, otherwise we start the thinning pass (pseudo code is shown in Algorithm 3). In the thinning pass we attempt to collapse every edge which is not a boundary one, does not connect two boundary vertices and fulfills the edge collapse feasibility condition. We perform the collapse only if it improves the quality locally or if the quality of the affected tetrahedra after the operation is not smaller than a threshold value $q_0 = 0.5$. If the thinning pass improved the quality of the mesh sufficiently, we start the loop again, otherwise we record that the sequence of smoothing, topological and thinning passes did not manage to improve the quality of the mesh. If that happens three times in a row, the algorithm stops.

3.5 Tests and Results

We tested our schedule on the following meshes:

- Boid, Teapot and Deer are Delaunay tetrahedralizations generated by TetGen [89] with extremely bad dihedral angles due to the lack of interior vertices.
- Rand1 – used by Freitag and Ollivier-Gooch and also by Klingner and Shewchuk to evaluate their mesh improvement algorithms.

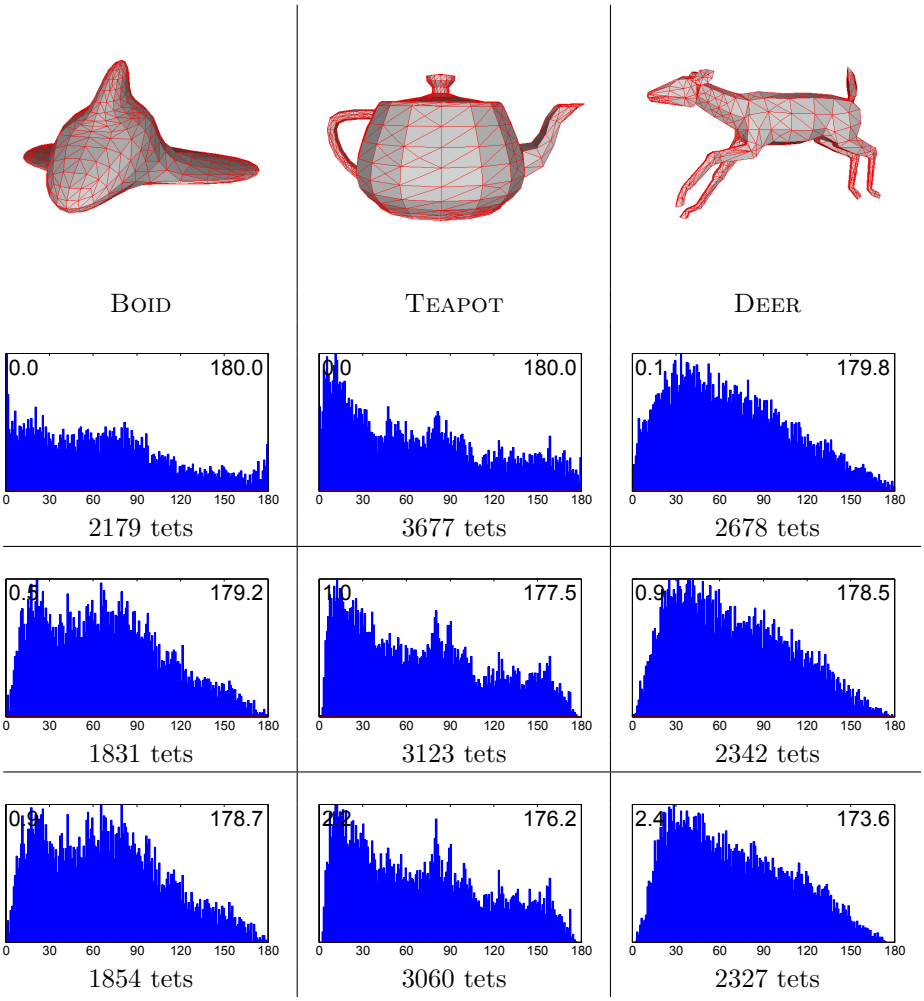

Table 3.1: Mesh quality improvement results for meshes: Boid, Teapot and Deer. Minimum biased sine measure was used for the first two and minimum sine quality measure was used for the last one. Pictures in the first row show the initial surface geometry of our meshes. Surface geometry remains the same after the mesh improvement, although the tesselation might change in flat regions. Histograms show, from the top to the bottom, the distribution of all dihedral angles in the original mesh, mesh improved without MFRT and mesh improved using MFRT.

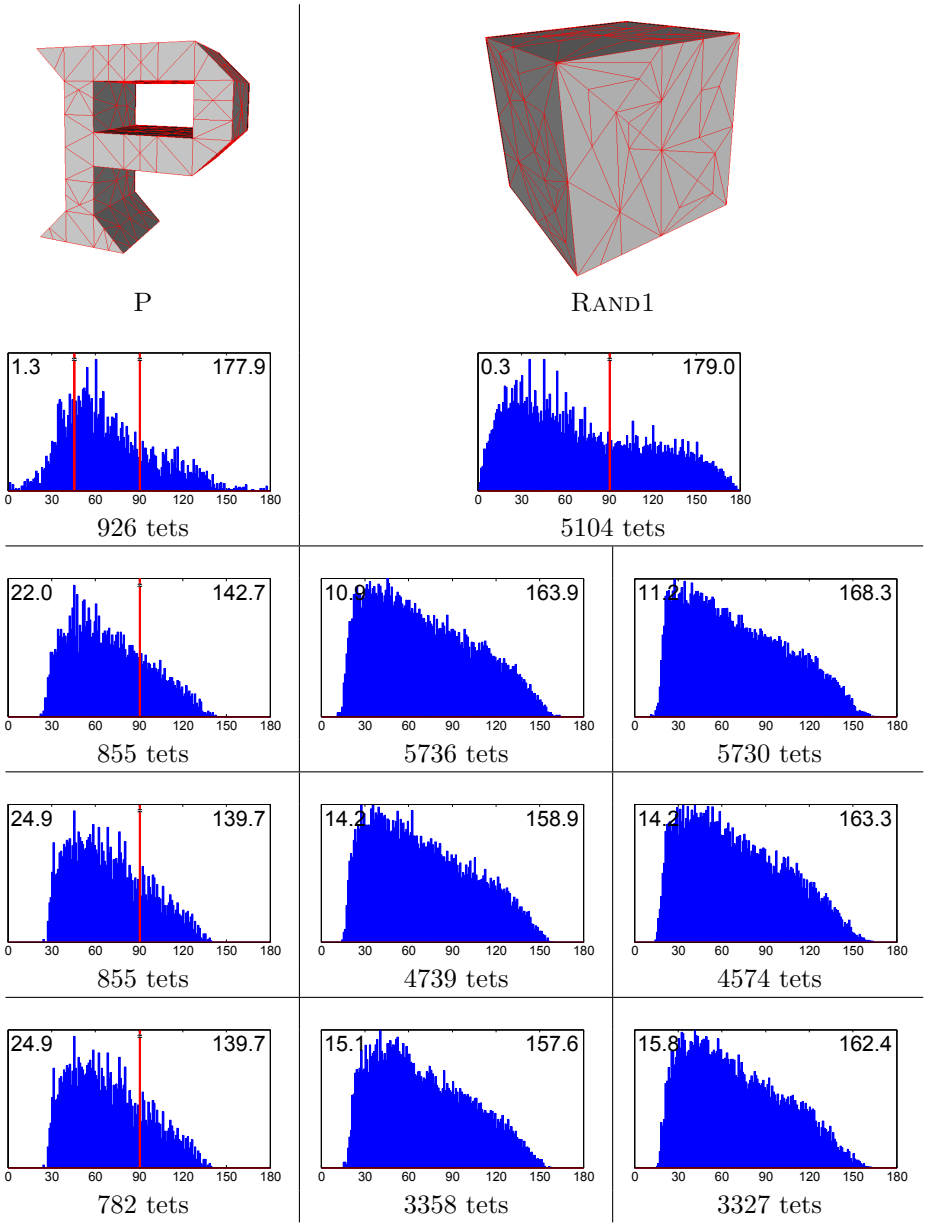

Table 3.2: Mesh quality improvement results for meshes: P (using minimum biased sine quality measure) and RAND1 (left column – using minimum biased sine quality measure, right column - using minimum sine quality measure). Histograms show, from the top to the bottom, the dihedral angle distribution in the original mesh, mesh improved without MFRT, mesh improved using MFRT, meshed improved using MFRT and thinning. Red bars indicate particularily abundant dihedral angles and were scaled down to increase the readability of the histograms.

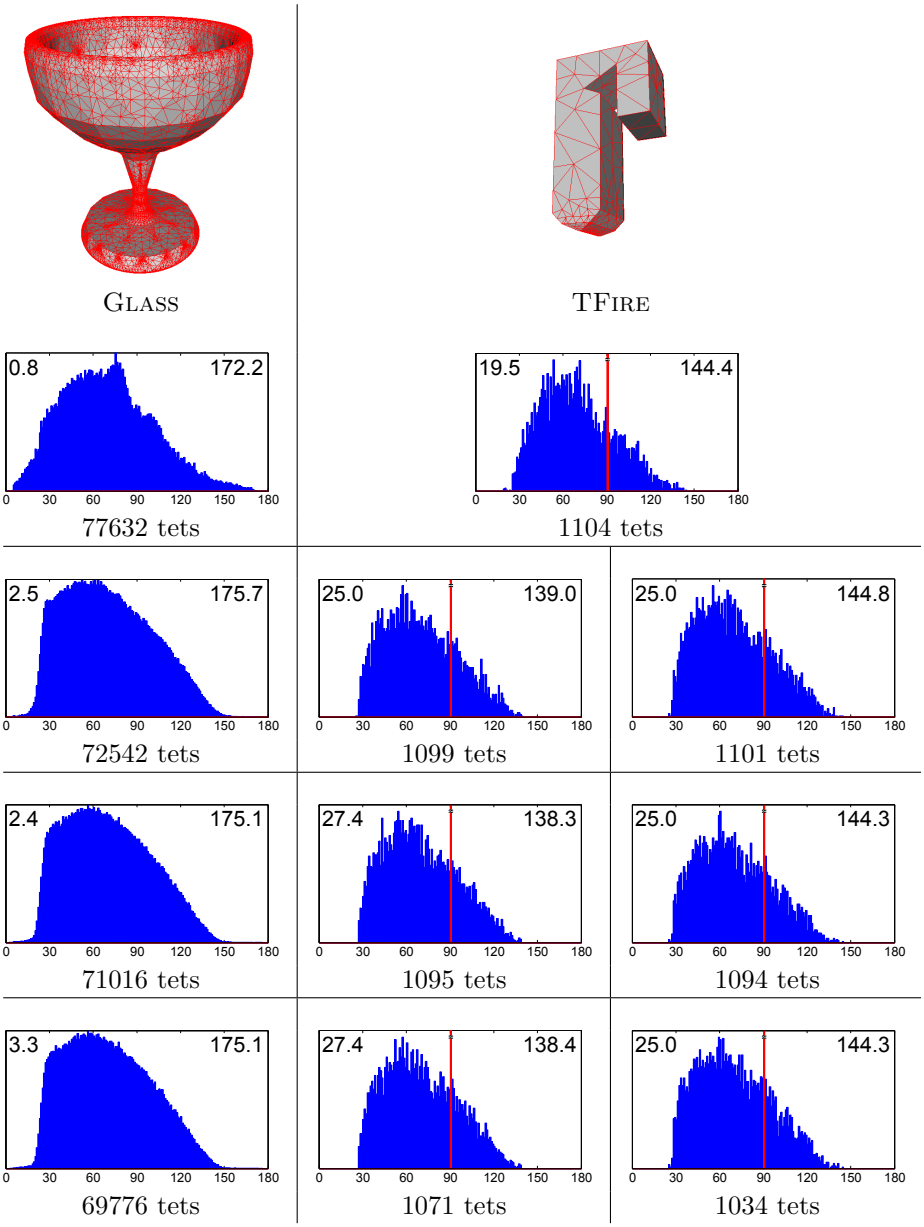

Table 3.3: Mesh quality improvement results for meshes: GLASS (using minimum biased sine quality measure) and TFIRE (left column – using minimum biased sine quality measure, right column - using minimum sine quality measure). Histograms show, from the top to the bottom, the dihedral angle distribution in the original mesh, mesh improved without MFRT, mesh improved using MFRT, meshed improved using MFRT and thinning.

Algorithm 1 IMPROVE(M)
{M is a mesh}

```
1: Smooth each vertex of M.
2: TopologicalPass(M)
3: failed ⇐ 0
4: while failed < 3 do
5:     Smooth each vertex of M.
6:     if M improved sufficiently then
7:         failed ⇐ 0
8:     else
9:         TopologicalPass(M)
10:        if M improved sufficiently then
11:            failed ⇐ 0
12:        else
13:            ThinningPass(M)
14:            if M improved sufficiently then
15:                failed ⇐ 0
16:            else
17:                failed ⇐ failed + 1
18:            end if
19:        end if
20:    end if
21: end while
```

Algorithm 2 TOPOLOGICALPASS(M)

```
1: Create the list of all tetrahedra in M.
2: for each tetrahedron t on the list, that still exists in M do
3:     for each edge e of t (if t still exists) do
4:         Attempt to remove edge e.
5:     end for
6:     for each face f of t (if t still exists) do
7:         Attempt to remove face f by first using multi-face retriangulation
           followed by multi-face removal.
8:     end for
9: end for
```

- P and TFIRE – used by Klingner and Shewchuk to evaluate their mesh improvement algorithm.

- GLASS – medium size mesh generated using TetGen [89] with few interior vertices and low quality boundary triangles.

Algorithm 3 ThinningPass(M)

```
1: for each edge e ∈ M that is not on the boundary do
2:     if e still exists then
3:         Find the vertices a and b of e.
4:         if b is not a boundary vertex then
5:             Attempt to collapse edge e: a ← b.
6:             if success then
7:                 Smooth a.
8:             end if
9:         end if
10:        if a is not a boundary vertex and e still exists then
11:            Attempt to collapse edge e: b ← a.
12:            if success then
13:                Smooth b.
14:            end if
15:        end if
16:    end if
17: end for
```

Unfortunatelly, Klingner and Shewchuk published the results of mesh improvement without vertex insertion only for a very few meshes, so the possibility of comparing our results to theirs was limited.

The results of mesh improvement for those meshes are presented in the Tables 3.1, 3.2 and 3.3. For the Boid mesh we tried to maximize the minimum biased sine quality measure for this mesh. The boundary of the mesh is nowhere flat so smoothing and topological operations are not allowed on the boundary. There are no interior vertices, so in fact smoothing and thinning cannot take place at all, as they would alter the surface geometry. Not much improvement can be achieved without vertex insertion in this case, but still we can see that the topological pass with MFRT is significantly more effective at fighting the worst dihedral angles than the topological pass without MFRT. The situation and the results are similar in the case of the Teapot mesh. We also obtain a significant extra improvement (6.4°) by the use of MFRT for the Deer mesh, while in this case we tried to maximize the minimum sine quality measure.

For Rand1 the use of MFRT allows us to narrow the dihedral angles range by as much as 8° for both sine and biased sine quality measures. Additionally, edge collapse allows us to decrease the complexity of the meshes by almost 35% and to narrow the dihedral angles range by almost 3° for sine quality measure – ultimately we obtain 15.8°–162.4°, and by 2° for biased sine quality measure – ultimately we obtain 15.1°–157.6°. For comparison, the best results Freitag and

Ollivier-Gooch [34] obtained for the same mesh was 15.0°–166.7° for minmax cosine quality measure (and 12.5°-167.3° for sine quality measure). Mesh P also benefits significantly from the use of MFRT – it narrows the dihedral range by 6°, but in this case the thinning pass does not improve the extreme quality values.

The TFire mesh also benefit from adding the MFRT and the edge collapse operation, although not as significantly as the previous ones. Still, our result 24.9°–139.7° is better than 21.3°–147.1° obtained by Klingner and Shewchuk [52].

In case of the Glass mesh, we can notice that our mesh improvement algorithm actually expands the dihedral angles range. This is due to the lack of extremely obtuse angles in the original mesh, and due to the fact, that the mesh operations we use choose to "sacrifice" good quality tetrahedra in order to locally improve the worst tetrahedron. However, we can notice that the we still benefit from inclusion of MFRT and thinning in the mesh improvement algorithm.

3.6 Discussion and Future Work

Our results show that using the multi-face retriangulation operation alongside smoothing and topological operations from the previous works can lead to better improvement of the dihedral angles and should be included in the standard repertoire of the topological operations for tetrahedral meshes. For the meshes we tested, we obtained a narrowing of the range of dihedral angles by up to 8° without inserting a single Steiner vertex. Additionally, edge collapse can also improve the worst dihedral angles and decrease the complexity of the mesh by up to 35%, esspecially when the initial quality of the mesh is very poor.

However, during our experiments we have noticed that the mesh improvement algorithm is still prone to get stuck in the local minima, even if we use multi-face retriangulation – in a few cases, running the algorithm with some mesh operations "switched off" (for instance operations on the boundary of the mesh) leads to better results than running the algorithm with the full repertoire of mesh operations. This is, of course, a consequence of using a greedy, hill-climbing approach. This could possibly be improved by applying a randomized approach.

It is also important to notice that our algorithm is designed for valid input meshes. If the initial mesh has inverted tetrahedra the algorithm might fail to remove them. Also, the tetrahedron quality measures we used are not particularily well suited for meshes with inverted tetrahedra, since they lose continuity

as the tetrahedron gets inverted.

In the future we are going to further investigate the possibilities of mesh improvement without Steiner vertex insertion, also with other quality measures, such as the *volume-length* measure [76] V/l_{rms}^3, where V is the signed volume of a tetrahedron and l_{rms} is the root-mean-squared edge length.

Acknowledgments

We thank Frederik Gottlieb for the implementation of the core part of the data structure that we are using, Bryan Klingner, Jonathan Richard Shewchuk and Mads Fogtmann Hansen for meshes, geometric models and discussion. We would also like to thank Vedrana Andersen for helping us give this paper its final shape and anonymous reviewers for useful suggestions.

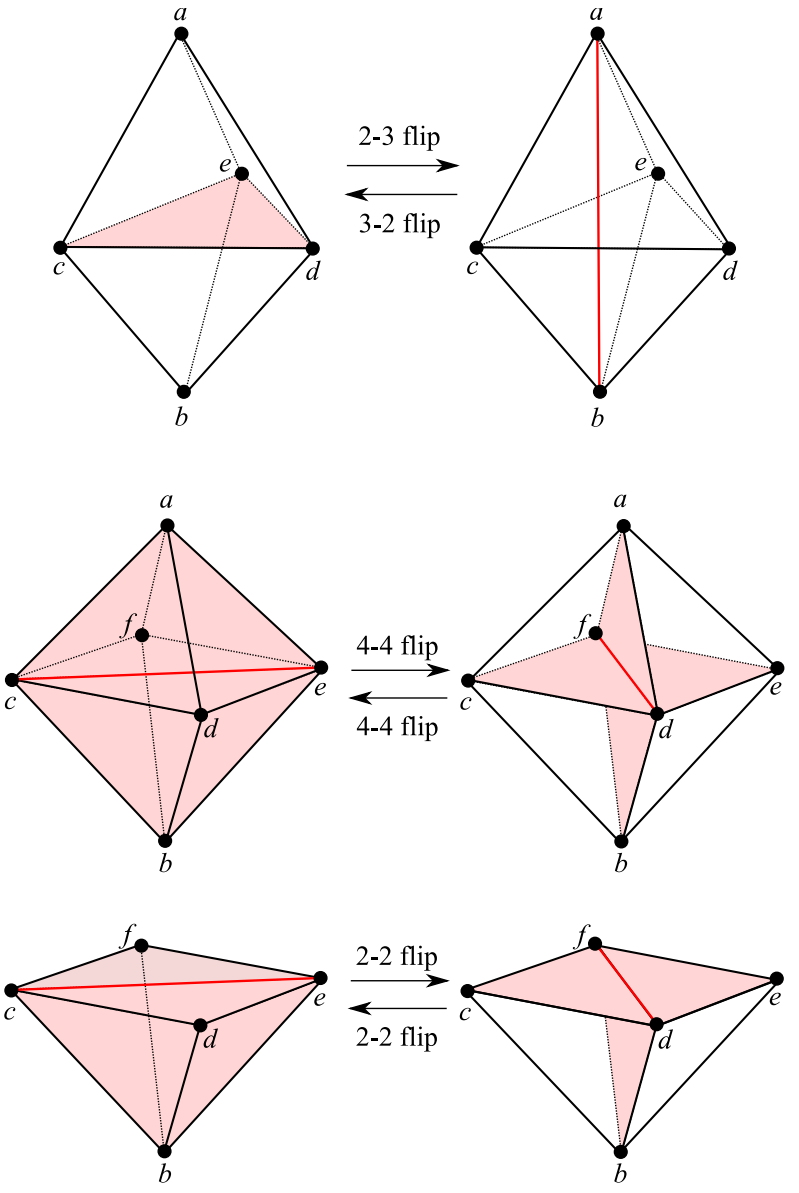

Figure 3.3: Simple topological operations.

Chapter 4

Deformable Simplicial Complexes

Marek Krzysztof Misztal, Technical University of Denmark
Jakob Andreas Bærentzen, Technical University of Denmark

Submitted to *ACM Transactions on Graphics*.

Abstract. We present a novel, topology adaptive method for deformable interface tracking, called the *deformable simplicial complexes* (DSC). In the DSC, the interface is represented explicitly as a piecewise linear curve (in 2D) or surface (in 3D) which is a part of a discretization (triangulation/tetrahedralization) of the space, such that the interface can be retrieved as a set of faces separating triangles/tetrahedra marked as *inside* from the ones marked as *outside* (so it is also given implicitly). This representation allows robust topological adaptivity and, thanks to the explicit representation of the interface, it suffers only slightly from numerical diffusion. Furthermore, the use of an unstructured grid yields robust space adaptivity. Also, topology control is simple in this setting. We present the strengths of the method in several examples: Simple geometric flows, fluid simulation, point cloud reconstruction and cut locus construction.

4.1 Introduction

Deformable interfaces are useful in a great many applications, such as fluid dynamics where we need to track the interface between fluids, 3D modelling where the interface is the surface of the object, and image analysis where such methods are used in segmentation and object recognition.

Topological adaptivity – meaning that interface components may split and merge – is crucial to many applications of deformable interface methods, and whenever this is the case, Eulerian methods, such as the well known *level set method* and its numerous variations are often employed. While many of these methods are powerful, they tend to work by sampling on a regular grid a function whose 0-level set represents the interface. This sampling, in turn, tends to introduce numerical diffusion and makes small details infeasible. In this project, it has, therefore, been our goal to avoid regular grids altogether but without loosing the most important advantages of the Eulerian outlook.

The main contribution of this paper is that we introduce deformable simplicial complexes, DSC, a generic method for tracking deformable interfaces which combines many of the advantages of Eulerian and Lagrangian mehods. In particular, the method suffers from little numerical diffusion, allows for transparent topology changes, and provides an explicit triangles mesh (in 3D) representation of the interface which changes only where needed between time steps. We investigate several applications: Fluid dynamics, point cloud reconstruction and cut locus construction.

4.2 Related Works

Traditionally, methods for deformable interface tracking fall into two categories: *explicit* (*Lagrangian*) and *implicit* (*Eulerian*). Traditional Lagrangian methods, such as active contours or snakes, use parametrisation of the interface and apply the deforming velocity field ($\mathbf{v}$) directly to the interface points ($\mathbf{p}$):

$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = \mathbf{v}(\mathbf{p}).$$

This approach leads to trouble once the topology of the interface changes. An efficient collision detection mechanism is needed to detect self-intersections of the interface, and once it happens, costly reparametrisation is needed, along with surgical cuts (as in [36], although in recent work by [13] this problem is mitigated by not allowing self-intersections). Those problems do not occur in Eulerian methods, such as the *level set method* (*LSM*, [75]). LSM represents a n-dimensional interface as the 0-level set of a $(n+1)$-dimensional function $u(x_1, \ldots, x_n, x_{n+1})$ (signed distance function is usually the choice), defined on the nodes of a regular grid. The evolution of the interface due to the velocity field $\mathbf{v}$ is then described by the following partial differential equation, also known as that *level set equation*:

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u = 0.$$

This approach provides trivial and robust topological adaptivity. However, the LSM also exhibits several drawbacks: it is bound to a certain scale, it suffers from significant numerical diffusion for features near the sampling rate irrespective of discretization, it does not allow explicit interface representation and it relies on calculations in one dimension greater than the interface itself.

In order to address the disadvantages of purely Lagrangian or Eulerian methods, several hybrid methods emerged. Some methods are based on triangle meshes, but use voxel grids to resolve topological changes. One of the earliest examples is the topologically adaptive snakes method by [64]: the interface is represented with a triangle mesh, but a voxel grid is used to resolve topological changes, leading to many of the issues of Eulerian methods, and moreover movement of the interface is restricted to pure expansion or contraction. In more recent work, [96] use a Lagrangian approach and voxel grid-based method only to locally resolve topological changes (and simplify complex areas). Since the changes are only local and as needed, the method is able to pass the Enright test [25] with no visible changes to the rotating geometry. However, the method does require building a signed distance field each time step and the scale of the voxel grid affects the results.

Some authors detect intersections using collision detection and resolve topology

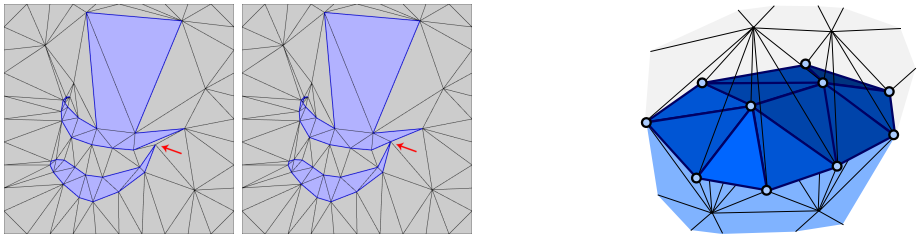

Figure 4.1: Interface representation in deformable simplicial complexes (2D on the left, 3D on the right). Exterior triangles (tetrahedra) are light gray, interior – blue. Simplices belonging to the interface (edges and vertices in 2D; faces, edges and vertices in 3D) are highlighted in dark blue. On the left, the red arrow indicates where topology changes take place. Note also the difference in scale between the largest and the smallest triangles.

changes using mesh transformations. [55] propose a method where violation of an edge length constraint indicates an intersection or self-intersection, and a so-called axial transformation is used to create a tunnel if two components merge, and, if a component splits, an annular transformation is used to disconnect the pieces. [99, 100] propose a method where self-intersections are removed from an evolving triangle mesh at each time step using a method that greatly resembles Boolean operations on meshes.

The work most directly related to ours is the method due to [79]. The authors proposed a method which is based on a triangle mesh representation of the interface, but once the vertices have been moved, a restricted Delaunay tetrahedralization of the interface is performed. A test is performed on each of the new tetrahedra in order to label them as interior or exterior. If a vertex is found to be shared only by identically labeled tetrahedra, it is removed. This method shares a number of advantages with our method. In particular, it can be extended to multi-phase simulations, and it suffers only little from numerical diffusion, but there is no detection of what happens between time steps. Arguably a small object could pass through a thin wall if the time step was not properly tuned, and the precise points where interface collisions occur are not detected. Lastly, it would be difficult to extend their method to do topology control which is simple with our approach.

4.3 Deformable Simplicial Complexes

Like the level set method [75], DSC is a method for dealing with deformable interfaces. In DSC, the interface is represented explicitly as a set of faces of simplices belonging to a simplicial complex one dimension higher. These simplices belong either to the object or the exterior. Simplices never straddle object boundaries. Thus, in 2D, the computational domain is divided into triangles, and the deforming interface is the set of line segments which divide interior triangles from exterior triangles. Similarly, in 3D, the interface is the set of triangles dividing interior tetrahedra from exterior tetrahedra. Both the 2D and 3D case are illustrated in Figure 4.1.

The interface deformation is performed by moving the vertices, and this means that the method preserves the advantages of the Lagrangian methods: It suffers from little numerical diffusion, and there is an explicit representation of the interface which, furthermore, does not change *gratuitously* between time steps. Moreover, the simplicial complex does not have to be regular meaning that we can allow details of significantly different scale in the same grid (c.f. Figure 4.1 left).

On the other hand, our approach also shares what we perceive as the biggest advantage of the Eulerian methods. Whenever the interface moves, the triangulation is updated to accommodate the change. If two different interface components collide, this change causes them to merge. Thus, topology is allowed to change transparently to the user—although with our method it is also possible to disallow topological changes.

4.3.1 Method description

We will first describe the DSC method in 2D. The basic idea is to compute the new position for the interface vertices in each time step (using a user defined velocity function) and then attempt to displace the interface vertices to the new positions, one after another. If the displacement of a vertex does not invert any triangle in its star (*1-ring*) then it is performed. Otherwise, it is moved in a straight line as far as possible. When all vertices have moved, we perform a *mesh improvement* routine and the displacement of the vertex to its final position is continued in the next substep, taking as many substeps as required to reach the end of the time step.

The outline of the DSC algorithm (both 2D and 3D version) is given by the following pseudocode:

Algorithm 4 DeformableSimplicialComplexes($M, \mathbf{v}$)
{M is a triangle/tet mesh conforming to the initial interface}
{$\mathbf{v}$ is a velocity function for the interface vertices}

```
1:  t ⇐ 0
2:  while t < T do
3:      for each marked (interface) vertex p_i do
4:          compute final vertex position p̃_i ⇐ p_i + v(p_i) · Δt
5:      end for
6:      complete ⇐ false
7:      counter ⇐ 0
8:      while not complete and counter < 10 do
9:          complete ⇐ MoveVerticesStep(M, {p̃_i})
10:         improve mesh quality
11:         counter ⇐ counter + 1
12:     end while
13:     t ⇐ t + Δt {Δt is a time-step}
14: end while
```

Algorithm 5 MoveVerticesStep($M, \{\tilde{\mathbf{p}}_i\}$)
{$\{\tilde{\mathbf{p}}_i\}$ is a set of the new positions of the interface vertices}

```
1:  complete ⇐ true
2:  for each marked vertex p_i do
3:      if ‖p_i − p̃_i‖ > 0 then
4:          compute the intersection t_0 of the ray p_i + t · (p̃_i − p_i) with the
            link of the vertex p_i
5:          if t_0 > 1 then
6:              p_i ⇐ p̃_i
7:          else
8:              move the vertex p_i to the intersection point p_i + t_0 · (p̃_i − p_i)
9:              complete ⇐ false
10:         end if
11:     end if
12: end for
13: return complete
```

The mesh improvement step (Algorithm 4, Step 10) aims at improving the quality of the mesh in order to decrease the likelihood of situations where the displacement of a vertex to its final position is not possible and to remove the degenerate triangles created when such situations occur. The following operations are used in the 2D mesh improvement routine:

- *Mesh quality improvement:* Moving vertices tends to introduce degenerate triangles (tetrahedra) and, in general, it almost invariably reduces the quality of the simplices. Consequently, we need to improve the quality – both because poorly shaped triangles (tetrahedra) are the most likely to be inverted as we move the interface vertices (and hence we cannot displace them in one go) and also because we often want to use the mesh as a computational grid and the quality of elements might significantly affect the accuracy of the results. We perform Laplacian smoothing of the non-interface vertices. Edge flips are performed for all non-Delaunay edges in the mesh which are not interface edges.

- *Interface topology changes:* Edge flips are also performed when an interface edge is the longest edge of a *cap* (nearly degenerate triangle with its obtuse angle greater than a certain threshold value θ_{cap}, $\cos\theta_{cap} \approx -1$) and if the vertex of the cap opposite to this edge (*cap tip*) lies on the interface as well; the newly created triangles are labeled according to the other triangle adjacent to the flipped edge (see Figure 4.1 left).

- *Detail control:* In order to make the mesh improvement effective, we need some *degrees of freedom* – extra non-interface vertices, also known as "Steiner" vertices. However, we must also make sure that we do not introduce too many vertices, otherwise the complexity of the mesh might grow dangerously high. Non-interface edges are removed if this can be done without changing the interface and without introducing degenerate triangles or triangles with a minimum angle smaller than a given threshold. This also removes a non-interface vertex. We add non-interface "Steiner" vertices by inserting vertices in the barycenters of needles (triangles with one extremely small angle). This will produce two triangles with even smaller angles, but these are removed by edge flips.

- *Degeneracy removal:* Whenever we move a vertex as far as it is possible without inverting the triangles in the mesh, we introduce degenerate triangles (area of which is close to 0). Hence we have to remove triangles smaller than a certain threshold area $a_{degenerate}$ or with one of the angles smaller than a certain threshold angle $\theta_{degenerate}$ through edge collapse of the shortest angle (if it produces a valid mesh).

4.3.2 3D deformable simplicial complexes

The outline of the 3D version of DSC follows the main steps of the 2D algorithm. However, some of the tools useful in the 2D case exhibit mediocre performance in the 3D. Laplacian smoothing may produce inverted tetrahedra in 3D despite working quite well in 2D. Likewise, while in the 2D case Delaunay meshes are

usually high quality, in 3D they often contain numerous, nearly-flat tetrahedra (called *slivers*), which easily get inverted due to a small displacement of their vertices and introduce significant errors in finite element computations [88].

To overcome the first difficulty, we use *smart Laplacian smoothing* (moving a vertex towards the barycenter of its neighbors only if it improves mesh quality locally) supported by L. Freitag's *optimization based smoothing*, [33]. Also, Delaunay tetrahedralization is used only to create the initial tetrahedralization of the domain (using Tetgen [89]), which then undergoes mesh improvement (analogous to the mesh improvement algorithms described in [34] and [52]). Instead of using the Delaunay quality measure of a tetrahedron (minimum solid angle), we aimed for a quality measure which would penalize both slivers and long, needle-shaped tetrahedra, characterized by near-zero volume despite the fact that the distances between their vertices (edge lengths) might be large. We decided to use the volume-length ratio [76]:

$$Q(\sigma) = 6\sqrt{2}\frac{V(\sigma)}{l_{\text{rms}}^3},$$

where $V(\sigma)$ is the oriented volume of a tetrahedron t, and l_{rms} is the average (root-mean-squared) edge length:

$$l_{\text{rms}} = \sqrt{\frac{l_{12}^2 + l_{13}^2 + l_{14}^2 + l_{23}^2 + l_{24}^2 + l_{34}^2}{6}}.$$

This function is scale-indifferent and measures how different a tetrahedron is from the regular tetrahedron (it equals 1 for the regular tetrahedron, and it is close to 0 for slivers and needles). This is of course not the only quality measure having those properties (for other options see [40, 88]), we chose it for its simplicity and smoothness.

4.3.2.1 3D Mesh Improvement

Analogously to the 2D case, the tetrahedral mesh used in the 3D DSC needs to be improved after each step of the deformation. The following operations are used in the mesh improvement step:

- *Mesh quality improvement:*
  - Smart Laplacian smoothing of the non-interface vertices and *optimization-based smoothing*, which moves the vertex in a way that maximizes the minimal quality of the tetrahedra in its coboundary (this is clearly a non-smooth optimization problem, for the details see [33]).

  - Likewise, we perform topological operations on the 3D meshes (generalizations of the *edge flip* in the 2D case): *edge remove*, *multi-face remove*, *multi-face retriangulation* (for more detail see Chapter 3, and also [34, 52, 68]) if they improve minimal quality locally.

- *Interface topology changes:* Topology changes occur when vertices from one component of the interface touch another part of the interface. In this case the two interface parts will be separated only by one or more degenerate tetrahedra. Those tetrahedra are either re-labelled (switched from *inside* to *outside*, or the other way round) or removed as a part of the degeneracy removal (discussed below).

- *Detail Control:* Non-interface edges are collapsed if it does not produce degenerate nor inverted tetrahedra, if it does not alter the interface and if it does not create tetrahedra of quality smaller than a certain threshold value $q_{collapse}$ after the operation. Edges which do not belong to the interface are split if the edge connects the domain boundary with the interface or two interface vertices and is longer than a certain threshold value l_{split}. This ensure the interface has enough freedom to move.

- *Degeneracy removal:* The topology of the interface changes when vertices from one part of the interface collide with another part of the interface. This introduces degenerate tetrahedra, and when these are removed as described below, the topology changes occur. **Tetrahedra**: If a tetrahedron's vertices are nearly coplanar (if the tetrahedron's minimum dihedral angle is smaller than a value $\theta_{dihedral}$) its largest face is found and a tetrahedron removal strategy is chosen accordingly to the position of its opposite vertex. **Faces**: If a face is a cap with its obtuse angle greater than θ_{cap}, the opposite edge is split and the edge connecting the new vertex and the cap tip is collapsed; if the face is a needle with smallest angle smaller than a threshold value θ_{needle}, the edge opposite to this angle is collapsed (both collapses are performed if they produce a valid mesh). **Edges**: Finally, we collapse edges shorter than a certain threshold value l_{min} if it produces a valid mesh.

Balancing edge collapses and edge splits is crucial for the optimal performance and the robustness of the method. While we are trying to keep the complexity of the embedding mesh reasonable (ideally proportional to the number of triangles in the interface mesh), we try to avoid creating edges that directly connect the interface with the domain boundary or connect two different parts of the interface, as the resulting tetrahedral mesh is too inflexible (difficult to improve) and prone to contain degenerate tetrahedra or faces and removing those might lead to significant alteration of the interface.

4.3.2.2 Surface Mesh Improvement

The success of a Lagrangian method often depends on the quality of the triangulation of the interface [13], especially when the velocity field computation depends on the geometry of the interface. Moreover, the quality of the tetrahedralization benefits from better shaped triangles in the interface [18, 86]. However, even if we start with a high quality triangular mesh, it might quickly deteriorate as we advect the interface. To prevent this we include several interface improvement operations which do not significantly alter the geometry of the interface:

- *Null-space smoothing* by [44]: moving each vertex only in the null space of its local quadric metric tensor. This way it smoothing does not change the geometry of the interface mesh. However, in our implementation we allow *slight* changes to the geometry in smooth regions.
- *Edge flip* of the interface edges: this can be seen as a special case of the edge removal (swap) operation in the ambient tetrahedral mesh. We perform the edge flip if its adjacent triangles do not fulfill 2D Delaunay criterion, if it does not lead to creating inverted tetrahedra in the mesh, if the edge is not a feature edge and if the volume of the tetrahedron spanned by its adjacent triangles is sufficiently small.
- *Edge split* for edges longer than a certain threshold value.
- *Edge collapse* for non-feature edges shorter than a certain threshold value.

4.4 Applications

4.4.1 Simple Geometric Flows

The benchmark tests for deformable models are simple geometric flows: Rotation, mean curvature flow and offsetting (motion in the normal direction) [75].

4.4.1.1 Rotation.

Rotation with an angular velocity ω around an axis $\mathbf{e}$ is performed by multiplying the interface vertices' positions by a rotation matrix $\mathbf{R}(\mathbf{e}, \Delta\theta)$ (where $\Delta\theta = \omega \cdot \Delta t$, Δt is the time step) in every time-step. This simple geometric

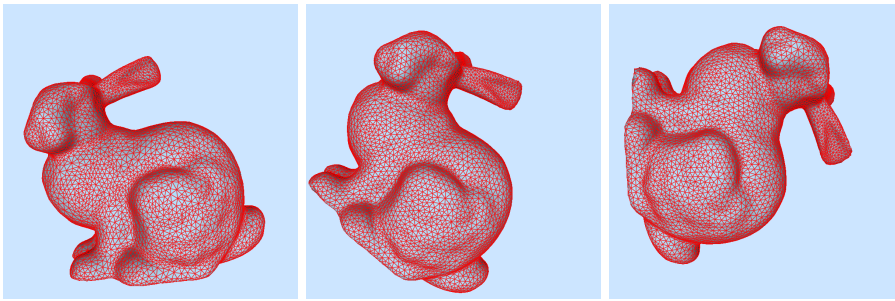

Figure 4.2: Rotation in the DSC framework. Changes to the interface mesh are minimal, mainly due to the surface mesh improvement operations.

flow poses a challenge for level set method, as the numerical diffusion causes the interface to lose sharp details and volume [75] and requires elaborate fixes to prevent it. Deformable simplicial complexes, like Lagrangian methods, does not suffer from such problems, as can be seen in Figure 4.2.

4.4.1.2 Mean Curvature Flow.

We compute the mean curvature of the interface using the *cotangent formula* [78]. The results are shown in Figure 4.3.

4.4.1.3 Offsetting.

Motion in the normal direction is performed using *face offsetting* [44]. Displacing the triangle mesh vertices by constant distance in the normal direction gives incorrect results. Instead, we offset the planes containing faces of the triangular mesh in the normal direction and find new vertices' positions from the intersections of these planes. The results are presented in Figure 4.4.

4.4.2 Cut Locus Construction

We utilize the possibility to preserve the topology of a front and to give the domain other topology than that of a disk in our cut locus construction method for Riemannian 2-manifolds, described in detail in [66]. The *cut locus* of a point $\mathbf{p}$ in a manifold $(\mathcal{M}, \mathbf{g})$ is essentially a set of all those points which are connected

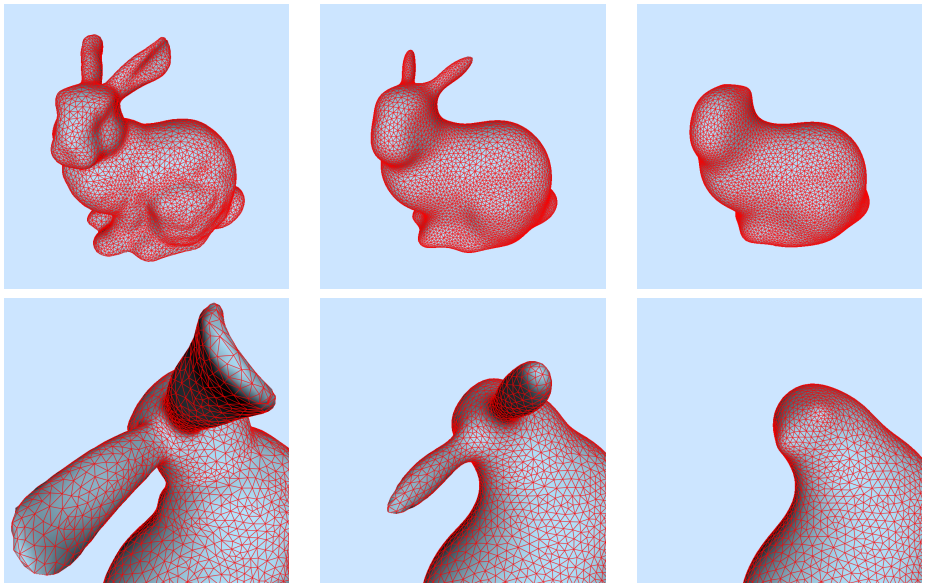

Figure 4.3: Mean curvature flow quickly erases high frequency details of the surface. A close-up of the ears in the lower row shows an important property of the DSC – the interface mesh only changes in the regions where it is needed, in order to accomodate the deformation; one can notice that most of the triangulation remains essentially unchanged between the frames, except for the ears, where the triangulation changes a lot (as this part of the mesh is affected the most by the mean curvature flow).

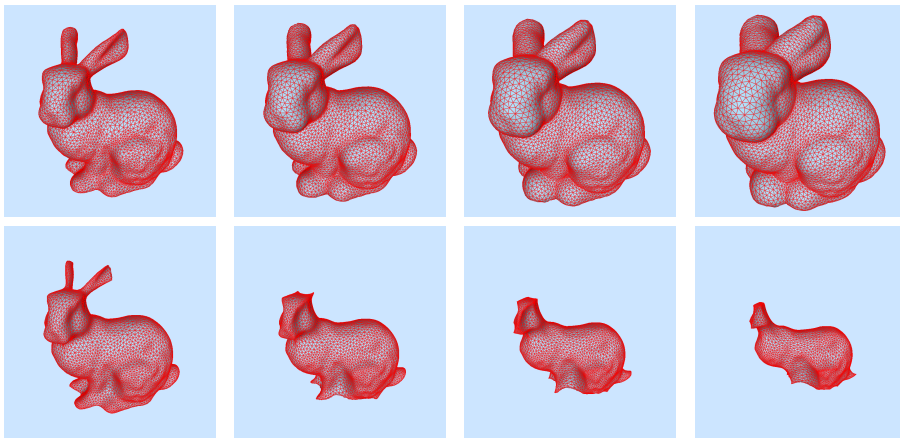

Figure 4.4: Interface offsetting: outwards (upper row) and inwards (lower row).

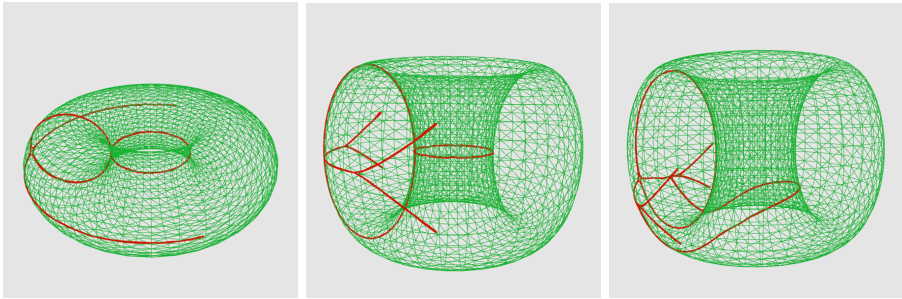

Figure 4.5: 2D DSC-based cut locus construction algorithm results for tori: Left – cut locus of a standard torus of revolution with circular generator $\mathbf{r}_1(u,v) = ((2+\cos(v))\cos(u), (2+\cos(v))\sin(u), \sin(v))$, for a point $(u,v) = (0,0)$; middle and right – cut loci of a stand torus of revolution with elliptic generator $\mathbf{r}_2(u,v) = ((2+\cos(v))\cos(u), (2+\cos(v))\sin(u), 2\sin(v))$, for points $(u,v) = (0,0)$ and $(0, 0.15\pi)$.

to $\mathbf{p}$ by more than one minimizing geodesic [82]. This can be also seen as a set of those points, where equidistance circles centered at $\mathbf{p}$ form cusps and self-intersect. Our algorithm utilizes the second approach. We advect the circle centered at $\mathbf{p}$ along the geodesics connecting its points with $\mathbf{p}$ with constant speed. The advection takes place in the coordinate chart (parametric domain, (u,v)-space) discretized using 2D deformable simplicial complexes and the 2D DSC mesh is given torus topology. Whenever the front is about to collide with itself (when this happens, degenerate triangles appear in the mesh) the interface stops. Some examples of cut locus construction results for tori are presented in Figure 4.5.

4.4.3 Point Cloud Reconstruction

In order to demonstrate space the adaptivity of the deformable simplicial complexes, we implemented a simple, topology adaptive point cloud reconstruction method. Our method vaguely resembles the algorithm proposed by Hoppe et al. [41], which is not topology adaptive. The initial interface is the triangulation of a bounding sphere of the point cloud. In every time step we compute a new desired configuration for each face f:

- if the vicinity of the face does not contain any point from the point cloud, we offset it in the normal direction, inwards;
- if the vicinity of the face contains points from the point cloud (and f is the

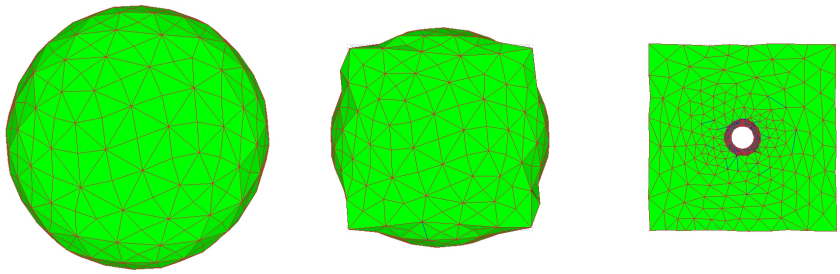

Figure 4.6: Reconstruction of an artificial point cloud – a box with a cylindrical tunnel.

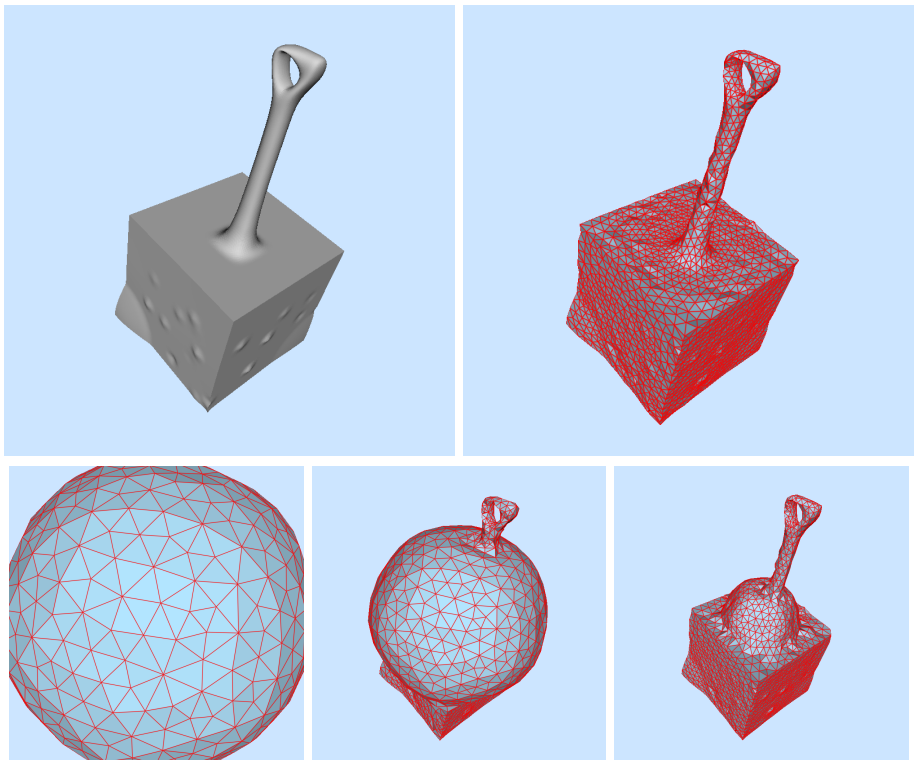

Figure 4.7: Upper row: The original data set (vertices of the mesh on the left, about 116k points) and the DSC-based reconstruction result. Lower row: A few steps of the reconstruction.

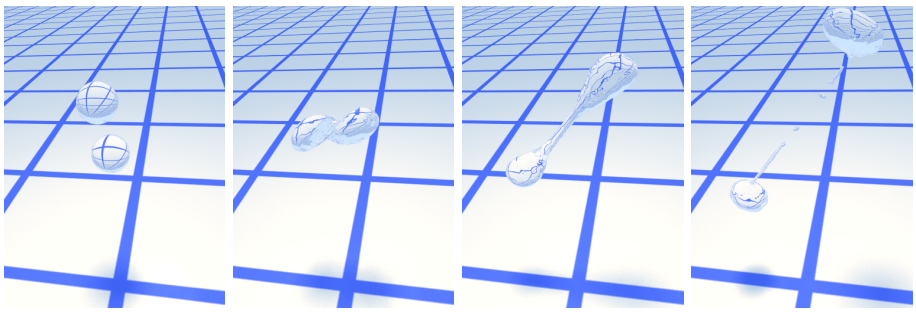

Figure 4.8: An example of fluid simulation using 3D example – two droplets colliding obliquely.

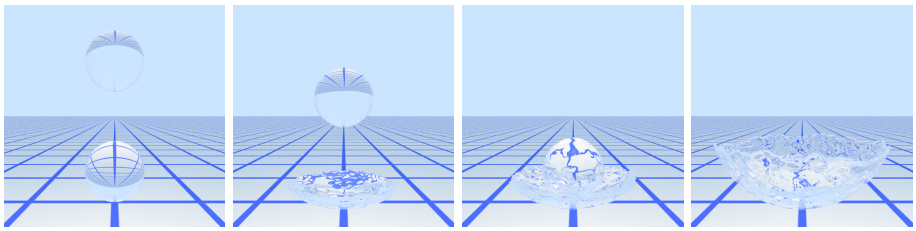

Figure 4.9: An example of fluid simulation using 3D DSC – two spherical volumes of fluid splashing inside a spherical vessel.

closest face for these points), we compute their least squares minimizing plane;

New vertices' positions are found from the intersections of those plane. Whenever the plane containing f does not approximate the points assigned to it accurately enough, we subdivide it. This extremely simple approach turns out to give decent results: it handles the sharp features correctly (unlike Poisson reconstruction [47] or early approached based on Radial Basis Functions [93] unless normal constraints are used [85]), and subdivision occurs only when needed (see Figures 4.6, 4.7).

4.4.4 Fluid Simulation

One of the main applications of deformable models is to track the free surface in fluid simulations. The free surface of a fluid undergoes drastic deformation and frequent changes in topology, so robust topological adaptivity is crucial.

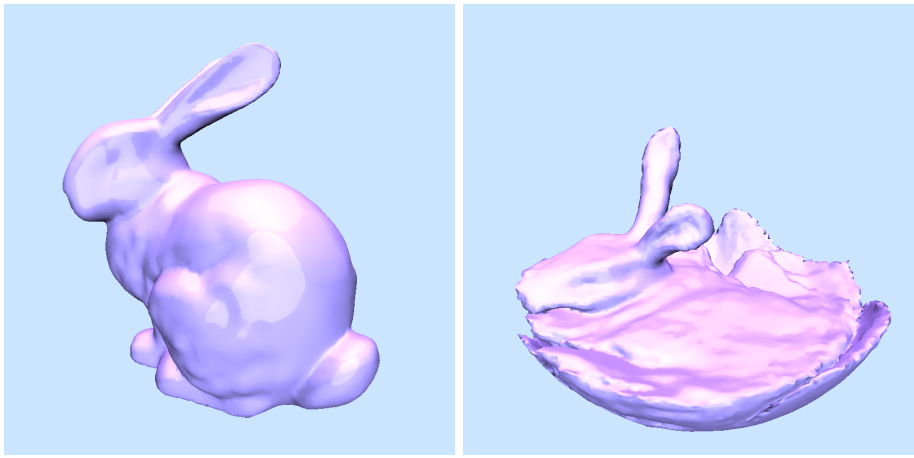

Figure 4.10: An example of fluid simulation using 3D DSC – Stanford bunny splashing inside a spherical vessel.

Traditionally, the level set method has been used alongside regular grid-based Navier-Stokes equation solvers [11]. This approach has proven to be extremely successful and is widely used in multiple practical application, but it has its limitations. Many of them, such as volume loss or difficulty handling curved solid boundaries have been addressed by numerous, often very elaborate patches. The lack of explicit interface representation makes it quite difficult to simulate surface phenomena and incorporating them in a fluid simulation framework requires fairly complex operations on the sub-grid scale meshes used for tracking fluid's surface [92, 97].

In [67] we present a completely new, finite element method-based approach to fluid simulation, utilizing the DSC mesh as an FEM mesh. Our solver rephrases incompressible Euler equations (inviscid flow equations) as an optimization problem with the surface energy term explicitly incorporated into the objective function. Besides that, the method automatically handles curved solid boundaries.

Some results are presented in Figures 4.8, 4.9, 4.10.

4.4.5 Scalability and Performance

The complexity of all the mesh improvement operations used in the 3D DSC algorithm is linear with respect to the number of tetrahedra in the mesh. Our

experiments suggest that also the number of tetrahedra in the DSC mesh is at most a quasi-linear function of the number of the interface triangles. Additional tests performed during the simple geometric flow examples have shown, that the DSC mesh is dominated by tetrahedra adjacent to the interface (out of approximately 80k tetrahedra, more than 70% are adjacent to the interface). The initial number of interface triangles in those examples was about 14k. In most of our experiments, the ratio of the total number of tetrahedra to the number of interface triangles fluctuates around the values 7-8.

Single iteration time might depend on the size of the displacement. If the displacements are large compared to the interface mesh edge size, several collisions might need to be resolved and bringing all interface vertices to their final positions might require more than just one or two executions of the main loop in Algorithm 4. On average, the time step seems to scale linearly with respect to the number of tetrahedra in the mesh. In the simple fluid examples, the size of the mesh is on the order of 10k and we could perform about 10 to 30 iterations per minute. In the Stanford bunny examples, the size of the DSC mesh is around 80k, and we could perform 1 to 4 iterations per minute, for reasonable displacement size (on 64-bit Intel® Xeon® CPU W5590 @ 3.33 GHz, 6 GB RAM).

4.5 Conclusions and Future Work

Our results show that deformable simplicial complexes can be an interesting alternative to the level set method. It shares the main advantage of the level set method: robust topological adaptivity, but is free of LSM's main drawbacks: significant numerical diffusion and poor scale adaptivity. Topology control is natural and simple in the DSC and the domain can have other topology than disk, as demonstrated by the cut locus application. Moreover, the DSC mesh can be used for finite element computations and provides fast collision detection mechanism.

The main drawback of the DSC in its current form is its speed. We were trying to optimize the performance while designing the method, however the in-depth analysis of it is still to be done. We believe that this way, and also by optimizing lower-level structures implementation, we could improve the performance significantly. Until then it is difficult to quantatively compare the DSC against other deformable interface tracking methods at this moment. For this reason we aimed at presenting the applicability of the DSC in the problems other methods have troubles dealing with. For the tasks requiring superior time performance, we would of course recommend simpler deformable models.

We would also like to continue working on the applications mentioned in Section 4.4 and explore the applicability of the DSC in another problems.

Acknowledgements

We are grateful to the following people for resources, discussions and suggestions: François Anton, Jeppe Revall Frisvad (Technical University of Denmark), Kenny Erleben (University of Copenhagen), Robert Bridson and Carl Ollivier-Gooch (University of British Columbia).

Chapter 5

Cut Locus Construction Using Deformable Simplicial Complexes

Marek Krzysztof Misztal, Technical University of Denmark
Jakob Andreas Bærentzen, Technical University of Denmark
Steen Markvorsen, Technical University of Denmark

Submitted to *Experimental Mathematics*.

Abstract. In this paper we present a method for appproximating cut loci for a given point p on Riemannian 2D manifolds. This is done by advecting a front of points equally distant from p along the geodesics originating at p and finding the lines of self-intersections of the front in the parametric space. This becomes possible by using *deformable simplicial complexes* (DSC, [65]) method for deformable interface tracking, since DSC provides a simple collision detection mechanism, allows for interface topology control and does not require the domain to have disk topology. We test our method for tori of revolution and compare our results to the benchmark ones from [38]. The method, however, is generic and can be easily adapted to construct cut loci for other manifolds of genera other than 1.

5.1 Introduction

5.1.1 Problem definition

The *cut locus* from a point $\mathbf{p}$ in a Riemannian manifold $\mathcal{M}$ is essentially a set of points in $\mathcal{M}$, which are connected to $\mathbf{p}$ by more than one minimizing geodesic. For example: in Euclidean space, the cut locus of a point is empty; on an n-sphere, the cut locus of a point consists of the point opposite to it (*antipodal* point); on an infinitely long cylinder, the cut locus of a point consists of the line opposite to that point. For a more mathematically rigorous definition, see [82].

Let us define the *distance circle* of a point $\mathbf{p}$ in a Riemannian manifold $\mathcal{M}$ and radius r as a set of points in $\mathbf{M}$ whose geodesic distance from $\mathbf{p}$ equals r. One can alternatively look at the cut locus of a point $\mathbf{p}$ as a set of points where the distance circle centered at $\mathbf{p}$ forms cusps and self-intersects as its radius increases.

Cut loci are among the main objects of study in differential geometry, with applications in diverse problems, like for example: recognition in computer vision [77] or forest fire modelling [35].

For the sake of clarity, in this paper we are primarily interested in 2-dimensional manifolds, embedded in $\mathbb{R}^3$, defined by a parametrization:

$$\mathbf{r} : U \to \mathbb{R}^3,$$

where $U \subset \mathbb{R}^2$. However, proposed method is applicable to any Riemannian 2-manifold, given a metric tensor. The method could also be extrapolated to 3-dimensional cases.

5.1.2 Torus setting

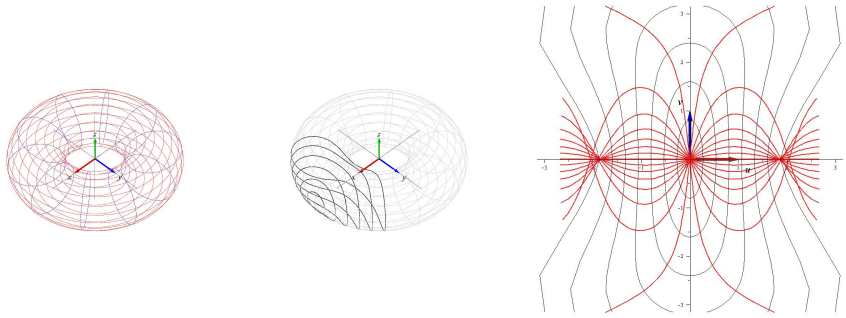

Figure 5.1: Standard torus with geodesics and equidistance circles in parameter space. The conjugate locus is clearly visible (2 places where the geodesics show first time reconvergence.)

The standard torus of revolution with circular generator in $\mathbb{R}^3$ is given by the following parametrization, where we have chosen specific values of diameter and tube radius:

$$\mathbf{r}(u_1, u_2) = ((2 + \cos(u_2)) \cos(u_1), (2 + \cos(u_2)) \sin(u_1), \sin(u_2)), \tag{5.1}$$

where $(u_1, u_2) \in \mathbb{R}^2$. Periodicity identifies points in the image. The parameter plane $\mathbb{R}^2$ is the universal cover. The fundamental domain is only, e.g. $(u_1, u_2) \in [-\pi, \pi] \times [-\pi, \pi]$.

The corresponding metric tensor is in this representation:

$$\mathbf{g}(u_1, u_2) = \begin{bmatrix} (2 + \cos(u_2))^2 & 0 \\ 0 & 1 \end{bmatrix}$$

If we work and operate in the parameter plane, when pushing a front forward to next level by a unit vector (a_1, a_2) at the position given by coordinates (u_1, u_2), then "unit" means that the length $\|(a_1, a_2)\|$ of (a_1, a_2) must be 1:

$$\|(a_1, a_2)\|^2 = (a_1, a_2)\mathbf{g} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = (a_1, a_2) \begin{bmatrix} (2 + \cos(u_2))^2 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = 1,$$

that is:

$$(2 + \cos(u_2))^2 a_1^2 + a_2^2 = 1.$$

So, if at the position (u_1, u_2) we want to go with unit speed in the *direction* given by the vector $(\cos(\theta), \sin(\theta))$, then in the (u_1, u_2)–plane we should go with the velocity vector given by:

$$\mathbf{V} = \frac{(\cos(\theta), \sin(\theta))}{\sqrt{(2 + \cos(u_2))^2 \cos^2(\theta) + \sin^2(\theta)}} \tag{5.2}$$

Similarly, if we want to go in a direction orthogonal to a given (front) vector (b_1, b_2) then the angle θ must be chosen, so that

$$(b_1, b_2) \begin{bmatrix} (2 + \cos(u_2))^2 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} = 0 \quad ,$$

which gives essentially two values of θ.

Going with velocity 1 away from the point $\mathbf{p} = (0, 0)$ along geodesics in a number of equally distributed directions gives Figure 5.1, where also the *distance circles* reached at given distances from $\mathbf{p}$ are indicated in a rough grid. The cut locus $\mathcal{C}(\mathbf{p})$ consists of those points where the distance circle front creates cusps and begin (locally) thereafter to cross itself together with those points where the distance circle front meets itself head-on and crosses itself.

5.1.3 Related work

The first generic tools allowing to numerically evaluate cut loci in Riemannian 2-manifolds began to appear in the early years of the last decade. R. Sinclair's and M. Tanaka's *Loki* [90] allowed to construct cut loci of genus 1 surfaces with very high accuracy (it was used in, e.g.: numerical evaluation of cut locus of a torus of revolution [38]). It is, however, very difficult to apply to surfaces of other genera, and its complexity is very high, even when a rough approximation of a cut locus is sought. J. Itoh and R. Sinclair developed *Thaw* [43] which addressed the problem of finding approximate ("quick and dirty" as the authors say) cut loci in a short time using a triangulation of a surface. It turned out to be an useful tool, giving reasonable results for almost arbitrary surfaces, but its computational cost increases exponentially with the level of subdivision of the triangular mesh and hence is not suitable for applications requiring higher accuracy. Neither of those tools is generalizable to solving a problem of finding cut loci in 3-dimensional Riemannian manifolds.

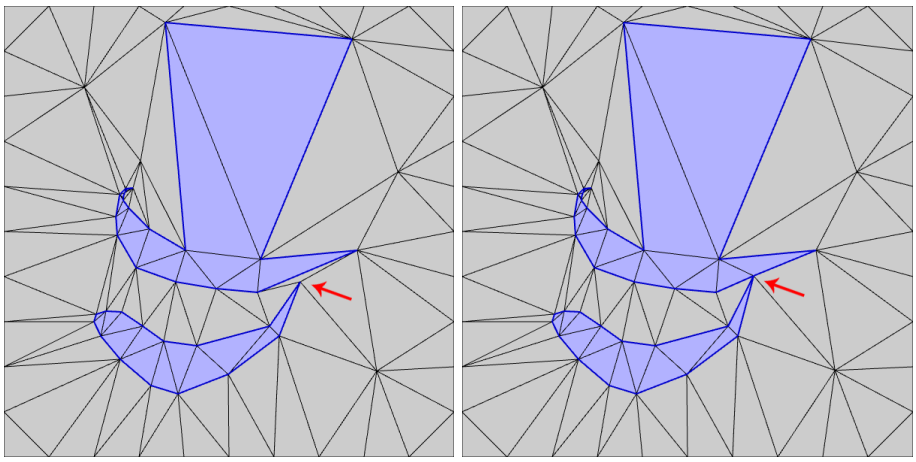

Figure 5.2: Interface representation in 2D deformable simplicial complexes. Exterior triangles are light gray, interior – blue. Simplices belonging to the interface (edges and vertices) are highlighted in dark blue. On the left, the red arrow indicates where topology changes take place. Note also the difference in scale between the largest and the smallest triangles.

5.2 Method Description

5.2.1 Deformable simplicial complexes

Like the level set method [75], DSC is a method for dealing with deformable interfaces. In DSC, the interface is represented explicitly as a set of faces of simplices belonging to a simplicial complex one dimension higher. These simplices belong either to the object or the exterior. Simplices never straddle object boundaries. Thus, in 2D, the computational domain is divided into triangles, and the deforming interface is the set of line segments which divide interior triangles from exterior triangles (as illustrated in Figure 5.2).

The interface deformation is performed by moving the vertices, hence it suffers from little numerical diffusion, and there is an explicit representation of the interface which, furthermore, does not change *gratuitously* between time steps. Moreover, the simplicial complex does not have to be regular meaning that we can allow details of significantly different scale in the same grid (c.f. Figure 5.2 left).

Whenever the interface moves, the triangulation is updated to accommodate the

change. If two different interface components collide, this change causes them to merge. Thus, topology is allowed to change transparently to the user—although it is equally simple to stop the interface and disallow topological changes. This is due to the fact that the DSC provides an intrinsic and immediate collision detection mechanism (whenever collision is about to happen, degenerate edges and triangles whose vertices are nearly collinear appear), which allows us to stop the deformation before the topological change occurs. We are going to utilize this *topology control* property in our cut locus construction algorithm.

The basic idea is to compute the new position for the interface vertices in each time step (using an arbitrary velocity function) and then attempt to displace the interface vertices to the new positions, one after another. If the displacement of a vertex does not invert any triangle in its star (*1-ring*) then it is performed. Otherwise, it is moved in a straight line as far as possible. When all vertices have moved, we perform a *mesh improvement* routine and the displacement of the vertex to its final position is continued in the next substep, taking as many substeps as required to reach the end of the time step.

The mesh improvement step aims at improving the quality of the mesh in order to decrease the likelihood of situations where the displacement of a vertex to its final position is not possible, provided that no self-collision of the interface occurs. The following operations are used in the mesh improvement routine:

- *Mesh quality improvement:* We perform Laplacian smoothing of the non-interface vertices. Edge flips are performed for all non-Delaunay edges in the mesh which are not interface edges.

- *Detail control:* Non-interface edges are removed if this can be done without changing the interface and without introducing degenerate triangles or triangles with a minimum angle smaller than a given threshold. This also removes a non-interface vertex. Sometimes we also need to add non-interface "Steiner" vertices. We do so by inserting vertices in the barycenters of needles (triangles with one extremely small angle). This will produce two triangles with even smaller angles, but these are removed by edge flips.

DSC does not require the underlying mesh to have disk (or $[0,1]\times[0,1]$) topology. In fact, in our algorithm for genus 1 manifolds, we give it torus topology, which allows us to detect all self-intersections of the growing distance circles.

5.2.2 Cut locus construction overview.

We are given a 2-dimensional Riemannian manifold $\mathcal{M}$ defined by a parametrization

$$\mathbf{r} : U \to \mathbb{R}^3,$$

and a point on this manifold $\mathbf{p}$ (the cut locus generator). At every time step the advecting front is represented as a piecewise linear curve in the coordinate chart. For each vertex of the discretization of the front we store another piecewise linear curve connecting it with the cut locus generator $\mathbf{p}$, approximating the minimizing geodesic. In every time step, we advect the front along the geodesics by a distance $\mathbf{V} \cdot \Delta t$, where Δt is the size of a time step and $\mathbf{V}$ is the velocity vector in the parameter space (this way, the front moves by a distance Δt on the manifold; compare with Equation 5.2). We add new positions of the vertices to the discretizations of the minimizing geodesics connecting them with $\mathbf{p}$. However, due to the discretization error, those new curves might not be geodesic any longer. We have to re-compute the geodesic distances of the new vertex positions from $\mathbf{p}$ (using an algorithm described in Section 5.2.3, which also produces a minimizing geodesic curve from an input, piecewise linear path) and, if it does not equal to the desired value d (which is the radius of the distance circle at the current time step) – correct the position of the vertices. We iterate those steps until all new vertex positions have geodesic distance d from $\mathbf{p}$.

Having new vertex positions we displace the vertices in the DSC framework (see Figure 5.3). *Head-on* self-collision of the interface is detected by apperance of very short, non-interface edges connecting two interface vertices or *caps* (very flat, obtuse triangles, base of which is an interface edge, and the apex of which is an interface vertex), we then stop the motion of the interface, in the former case by marking both vertices of an edge as *locked*, in the latter – by splitting the base of a cap at the point closest to the apex, and marking both the apex and the new vertex as *locked*. *Local* self-collision of the interface is detected by the interface edge becoming tangent to the geodesics connecting its vertices with $\mathbf{p}$. In this case we mark both vertices of such an edge as *locked*.

It is possible to add new vertices to the interface by splitting an interface edge at half when e.g. its length in the coordinate chart is larger than a certain threshold value, or when the geodesic distance between its vertices is larger than another threshold value. For an initial approximation of a geodesic for a vertex introduced this way we can choose the average of the geodesics of its neighbors (of course, such an approximation might not be geodesic, so it has to be corrected the same way as described above). Once all vertices of the interface are locked, it gives us an approximation of the cut locus of the point $\mathbf{p}$ in manifold $\mathcal{M}$.

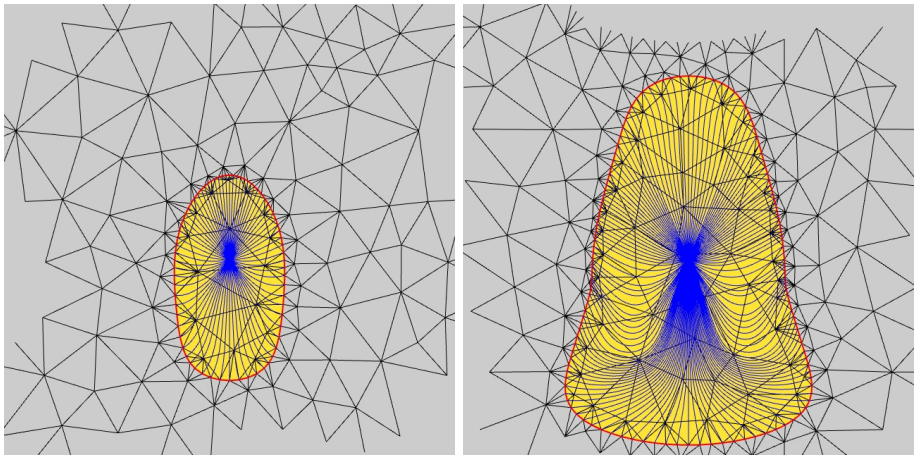

Figure 5.3: Growth of the distance circle in the DSC setup. The minimizing geodesics connecting the interface vertices with the cut locus generator are shown in blue.

5.2.3 Geodesic distance computation.

The details of the following method can be found in [61] and [4]. Let $\mathbf{c} : s \in [0,1] \to (x_1(s), x_2(s)) \in U$ be a smooth map. Then the curve $\mathbf{r} \circ \mathbf{c}$ is *geodesic* if it fulfills the following equations:

$$\frac{\partial^2 x_k}{\partial s^2} + \sum_{i,j} \Gamma^k_{i,j} \frac{\partial x_i}{\partial s} \frac{\partial x_j}{\partial s} = 0, \quad k = 1,2 \tag{5.3}$$

where $\Gamma^k_{i,j}$ are the second kind Christoffel symbols [2].

For a pair of points $\mathbf{x}^A = (x_1^A, x_2^A)$ and $\mathbf{x}^B = (x_1^B, x_2^B)$ in U we want to find a geodesic on $\mathcal{M}$ connecting $\mathbf{r}(\mathbf{x}^A)$ and $\mathbf{r}(\mathbf{x}^B)$. We solve the Equation 5.3 using the finite differences method. We approximate a geodesic as a piecewise linear curve $(\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^N)$, where $\mathbf{x}^0 = \mathbf{x}^A$, $\mathbf{x}^N = \mathbf{x}^B$ and $\mathbf{x}^p = \mathbf{c}(ph)$ for $p = 1, 2, \ldots, N-1$, where $h = \frac{1}{N}$. Then, by substituting:

$$\begin{aligned} \frac{\partial x_k}{\partial s}(ph) &\approx \frac{x_k^{p+1} - x_k^{p-1}}{2h}, \\ \frac{\partial^2 x_k}{\partial s^2}(ph) &\approx \frac{x_k^{p+1} + x_k^{p-1} - 2x_k^p}{h^2}, \end{aligned}$$

and multiplying by h^2 we obtain the following finite difference scheme:

$$x_k^p = \frac{x_k^{p+1} + x_k^{p-1}}{2} + \frac{1}{8}\sum_{i,j}\Gamma_{i,j}^k(\mathbf{x}^p)(x_i^{p+1} - x_i^{p-1})(x_j^{p+1} - x_j^{p-1}). \tag{5.4}$$

We can solve it by finding a fixed point of a transformation $H : U^{N-1} \to U^{N-1}$, transforming the approximation of a geodesic into a *better* approximation of a geodesic in Gauss-Seidel method iteration [3]:

$$H(\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{N-1}) = (\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^{N-1}),$$

where:

$$y_k^p = \frac{x_k^{p+1} + x_k^{p-1}}{2} + \frac{1}{8}\sum_{i,j}\Gamma_{i,j}^k(\mathbf{x}^p)(x_i^{p+1} - x_i^{p-1})(x_j^{p+1} - x_j^{p-1}).$$

We do that by finding a zero of $H(x) - x$, using Newton-Raphson algorithm [80]. Starting with an initial guess $(\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{N-1})$, we iterate:

$$\mathbf{Y}(1) = (\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{N-1}) \tag{5.5}$$

$$\mathbf{Y}(i+1) = \mathbf{Y}(i) - (\nabla_{\mathbf{Y}(i)}H - \mathbf{I})^{-1} \cdot (H(\mathbf{Y}(i)) - \mathbf{Y}(i)). \tag{5.6}$$

Given a good initial guess, the algorithm converges in a few iterations. This is, of course, the case in our algorithm, since we only make a small error as we move one step forward along the geodesic line, which was correct in the previous step.

5.2.4 Initialization

In order to obtain the initial front $\mathbf{L}_0$ of points whose distance from $\mathbf{p}$ equals a small value d_{init} we start with a piecewise linear approximation of a circle $\tilde{\mathbf{L}}_0 = \{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{N_0}\}$ centered at $\mathbf{p}$. The initial approximation of the geodesic connecting vertex $\mathbf{v}_i$ on the circle with $\mathbf{p}$ is a piecewise linear curve:

$$\tilde{\mathbf{Y}}_i = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{M_0-1}, \mathbf{u}_{M_0}\},$$

where:

$$\mathbf{u}_j = \frac{j}{M_0} \cdot \mathbf{p} + \frac{M_0 - j}{M_0} \cdot \mathbf{v_i}.$$

In order to push $\mathbf{v}_i$ onto the distance circle at d_{init}, we compute its geodesic distance d_i from p (as shown in Section 5.2.3), with $\tilde{\mathbf{Y}}_i$ as an initial guess. We then push it along the geodesic by a distance of $d^{init} - d_i$. We then iterate geodesic distance computation and vertex displacement until it lies on the desired distance circle.

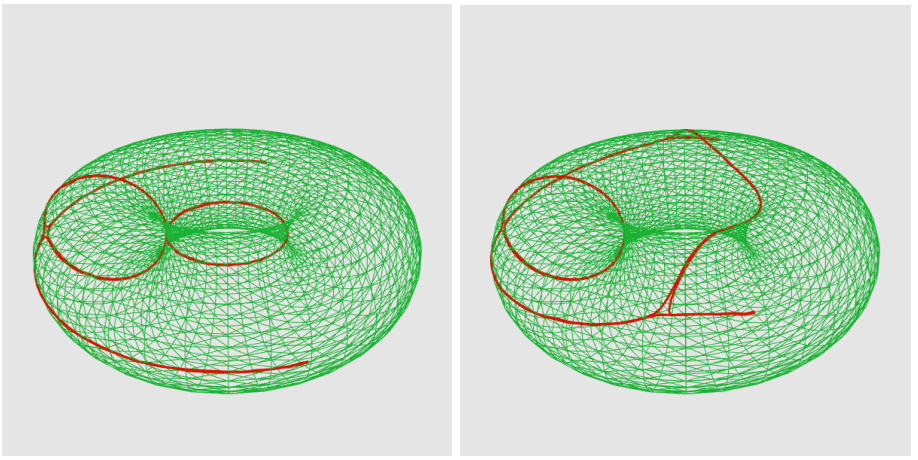

Figure 5.4: Cut locus construction results for $(u_1, u_2) = (0, 0)$ and $(0, 0.2\pi)$ (circular generator).

5.3 Tests and Results

5.3.1 Standard torus of revolution, circular generator

Cut loci of a torus defined by Eq. 5.1 for different positions of the point $\mathbf{p}$ on the generating circle are shown in Figures 5.4, 5.5.

5.3.2 Standard torus of revolution, elliptic generator

An elliptic cross section generator gives a different metric and a different geodesic spray. For a parametrization:

$$\mathbf{r}_\lambda(u_1, u_2) = ((2 + \cos(u_2))\cos(u_1), (2 + \cos(u_2))\sin(u_1), \lambda \sin(u_2)),$$

where $(u_1, u_2) \in [-\pi, \pi] \times [-\pi, \pi]$, $\lambda \in \mathbb{R}^+$, the corresponding metric is then:

$$\mathbf{g}_\lambda(u_1, u_2) = \begin{bmatrix} (2 + \cos(u_2))^2 & 0 \\ 0 & 1 + (\lambda^2 - 1)\cos^2(u_2) \end{bmatrix}.$$

The results for $\lambda = 2$ are shown in Figure 5.6.

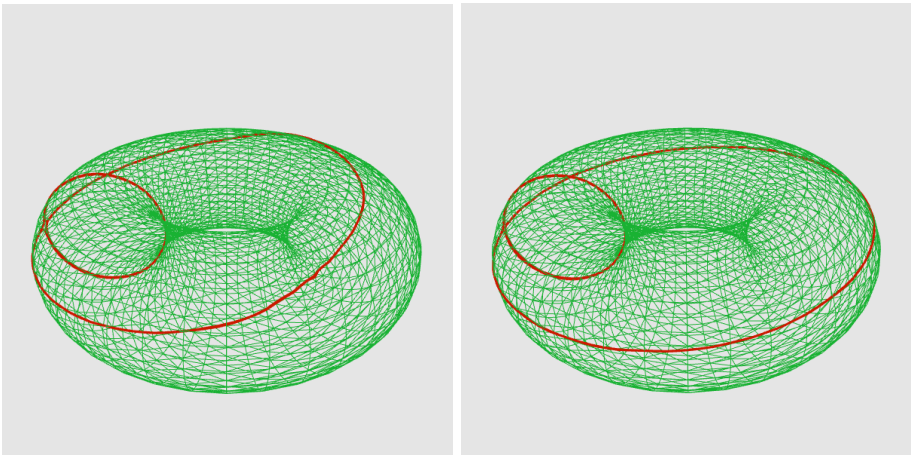

Figure 5.5: Cut locus construction results for $(u_1, u_2) = (0, 0.5\pi)$ and $(0, 0.9\pi)$ (standard torus of revolution, circular generator).

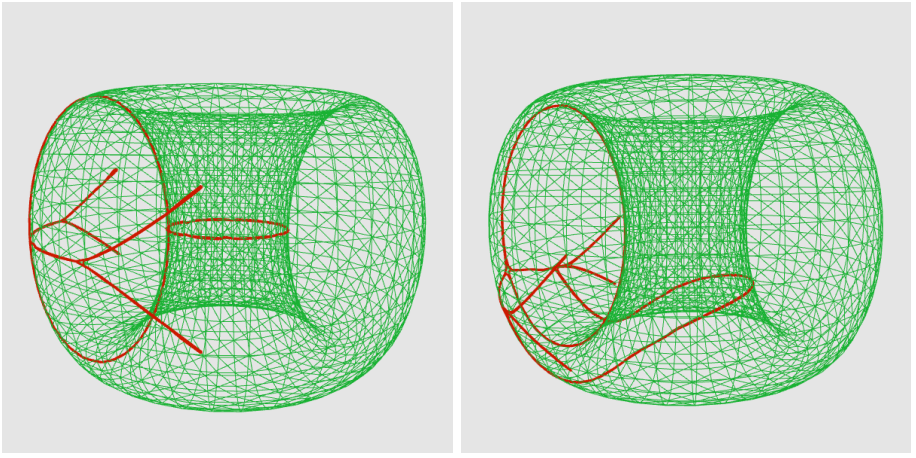

Figure 5.6: Cut locus construction results for $(u_1, u_2) = (0, 0)$ and $(0, 0.15\pi)$ (standard torus of revolution, elliptic generator).

5.3.3 Performance

Computation time in all our experiments was on the order of 10000 seconds (on Intel® Core™ Duo CPU T2350 1.86 GHz, 2 GB RAM).

5.4 Discussion

5.4.1 Accuracy and Complexity

The accuracy of our method depends on two factors: the level of subdivision of the advancing front and the size of the time step. In principle, the maximum error of the vertex position on the cut locus (in the parametric space) is bounded by the time step and maximum velocity in the coordinate chart:

$$\epsilon < \|\mathbf{V}\|_2 \cdot \Delta t.$$

The accuracy could be improved by computing exact collisions, which is not available in our implementation yet. We also believe, that even better results could be obtained, if more sophisticated interface refinement schemes are applied – for example, refinement based on the curvature of the front *in the manifold* – and we intend to investigate that.

The most costly operation in our algorithm is geodesic distance computation. Its cost is a quadratic function of the number of vertices in the piecewise linear approximation of the geodesic curve (because the matrix in the scheme 5.5 is banded). We perform a constant amount of geodesic distance computation calls per vertex in one time step. Since we add one vertex to the approximation of the geodesic curve in each time step, the size of it is inversely proportional to Δt. The time complexity of our algorithm is also proportional to the maximum number of vertices in the discretization of the front. Hence, the performance could be further optimized by, again, implementing a smarter front refinement scheme and by slower adding new vertices to the approximations of the geodesics (which are very densely sampled in the current implementation).

5.4.2 Generality of the method

We presented the results of our cut locus construction method for genus 1 Riemannian 2-manifolds, embedded in $\mathbb{R}^3$. However, careful reader might have

noticed that the method itself is generic, and can be easily adapted to handle other 2-manifolds. If a Riemannian 2-manifold $\mathcal{M}$ cannot be embedded in $\mathbb{R}^3$, it can alternatively be defined by three functions $E(u_1, u_2)$, $F(u_1, u_2)$ and $G(u_1, u_2)$ defining the metric tensor:

$$\mathbf{g}(u_1, u_2) = \begin{bmatrix} E(u_1, u_2) & F(u_1, u_2) \\ F(u_1, u_2) & G(u_1, u_2) \end{bmatrix}.$$

This representation is, of course, just as appropriate for our algorithm in its current form.

Our method also gives a possibility to track the evolution of the distance circles not in the parametric space U, but in the manifold itself. The DSC mesh would be a triangulation of the manifold $\mathcal{M}$ in this case. Most of the steps of the algorithm would not change in such a setting – however, we might need to use a different geodesic distance computation algorithm, e.g. [48] or some other approach to geodesic curvature minimization.

Since a 3D deformable simplicial complexes implementation exists [65], it would also be interesting and relatively straight-forward to use it in cut locus construction for Riemannian 3-manifolds.

Chapter 6

Optimization-based Fluid Simulation on Unstructured Meshes

Marek Krzysztof Misztal, Technical University of Denmark
Robert Bridson, University of British Columbia
Kenny Erleben, University of Copenhagen
Jakob Andreas Bærentzen, Technical University of Denmark
François Anton, Technical University of Denmark

Accepted to *Proceedings of VRIPHYS 2010: The 7th Workshop on Virtual Reality Interaction and Physical Simulation*, Copenhagen 2010.

Abstract. We present a novel approach to fluid simulation, allowing us to take into account the surface energy in a precise manner. This new approach combines a novel, topology-adaptive approach to deformable interface tracking, called the *deformable simplicial complexes method* (DSC) with an optimization-based, linear finite element method for solving the incompressible Euler equations. The deformable simplicial complexes track the surface of the fluid: the fluid-air interface is represented explicitly as a piecewise linear surface which is a subset of tetrahedralization of the space, such that the interface can be also represented implicitly as a set of faces separating tetrahedra marked as *inside* from the ones marked as *outside*. This representation introduces insignificant and controllable numerical diffusion, allows robust topological adaptivity and provides both a volumetric finite element mesh for solving the fluid dynamics equations as well as direct access to the interface geometry data, making inclusion of a new surface energy term feasible. Furthermore, using an unstructured mesh makes it straightforward to handle curved solid boundaries and gives us a possibility to explore several fluid-solid interaction scenarios.

6.1 Introduction

Since the mid-nineties of the previous century, fluid simulation has been extensively used in computer animated sequences of major motion pictures. Moreover, fluid simulation is important in many scientific applications, and simplistic fluid dynamics is even beginning to appear in real-time graphics applications.

Despite this rapid development, some problems remain with existing methods. Most existing methods are based on Eulerian simulations on fixed grids. These are prone to introducing gratuitous artificial viscosity in the simulations, volume loss, grid artifacts and, due to the lack of an explicit surface representation, surface tension is not easy to incorporate.

In the present paper, we propose a novel method for free-surface fluid simulation based on an irregular grid which dynamically adapts to the fluid volume. Our method is based on another recent technique for deformable interface tracking which we call *deformable simplicial complexes* (DSC, [65]). The gist of the DSC method is that the tracked surface is represented as a subcomplex (e.g. triangle mesh) of a simplicial complex (tetrahedral grid) which covers the entire computational domain. Tetrahedra are labelled according to which side of the interface they reside in, and the surface (interface) itself is the set of faces shared by a pair of tetrahedra belonging to opposite sides.

In this paper, we focus on how fluid simulation can be implemented on top of the DSC method, using the tetrahedral grid of the DSC method also as the computational grid for the fluid simulation. Our new method has at least two important benefits compared to previous methods:

- Because the grid changes only little to adapt to the changes in the water volume, we have very little numerical diffusion.
- Since we have an explicit representation of the water-air interface, it is very easy to add a surface energy term.

6.2 Related Works on Fluid Solvers

Many works are based on regular grids; as a general reference to grid-based incompressible flow in graphics we refer to the book [11]. Some of the foundations for grid-based works include an Eulerian approach to 3D fluid simulation [31] that demonstrated advantages over earlier work using particle systems, and 2D simulations and a semi-Lagrangian implicit time stepping method [91]. Recent work addresses irregular boundaries on grids [12].

Fluid animation on unstructured meshes, like our method, has been gaining popularity in the last five years. [27] simulated gases on static tetrahedral meshes to model the interaction of fluids with irregularly shaped obstacles, based on a finite volume method discretization of the divergence operator with a projection method to enforce incompressibility. On their staggered mesh only normal components of velocities are stored at the face centers, making it easy to apply solid boundary conditions. The main difficulty is the nontrivial reconstruction of the full velocity field from the face normal components. In comparison we use a finite element approach on a moving and deforming tethrehedral mesh and we store the full velocity vector at the vertices.

Deforming unstructured tetrahedral meshes were introduced in [28]. Here a moving mesh is used where the deformation is limited to preserve mesh quality. Our approach to advection uses the same generalized semi-Lagrangian method from this work. In comparison to our work we emphasize the topological operations needed when deforming the mesh.

Remeshing of the entire computational domain in each simulation step was used in [51]. The authors addressed two-way coupling of fluids and rigid bodies. Heuristics were used to generate high resolution meshes in visually important regions; our mesh refinement and improvement are based on mesh quality only.

(The coupling was later extended to deformable objects as well [17]; in our paper we do not address two-way coupling.)

Liquid simulation on unstructured tetrahedral meshes is presented in [16]. A semi-Lagrangian contouring method is used to extract the free surface and rebuild a tetrahedral mesh in every time step, and a body centered cubic lattice is used for the structure of the tetrahedral mesh. The liquid surface is embedded as a discrete submanifold in the tetrahedral mesh as in our case. However, rather than completely rebuilding a new mesh in each time step our approach is based on making local topological changes to remesh and improve mesh quality.

In [23] a new fluid simulation method is presented. A static staggered grid is used where velocities are stored at vertices and scalar fields at volume centers. The auhors apply a vorticity formulation of the Navier–Stokes equation whereas we use a momentum formulation. The authors employ discrete differential methods to guarantee a circulation-preserving flow, but do not handle liquids/free surfaces.

Another finite volume method is presented in [95]. Here full velocity vectors are stored at the face centers which add some problems to the pressure correction, necessitating an additional projection each time step.

While not strictly a fluid solver (focusing instead on elastoplastic materials) the work presented in [98] combines a highly detailed surface mesh with a non-conforming tetrahedral finite element simulator that makes frequent use of remeshing. In constrast we use a boundary-conforming tetrahedral mesh in our fluid solver.

In summary, past work is based on staggered meshes using face-centered velocity grid layouts. Most work on unstructured meshes deal with free surfaces using contouring and complete remeshing. Deforming meshes have been considered to control visual quality but in a deformation-limited manner; our approach follows the physical simulation and has no such limits. Further our work uses a finite element method for fluid simulation whereas previous work on fluid simulation on unstructured mehses use finite volume methods.

6.3 Deformable Interface Tracking

Traditionally, methods for deformable interface tracking fall into two categories: *explicit* (*Lagrangian*) and *implicit* (*Eulerian*). Traditional Lagrangian methods, such as active contours or snakes, use parametrisation of the interface and apply

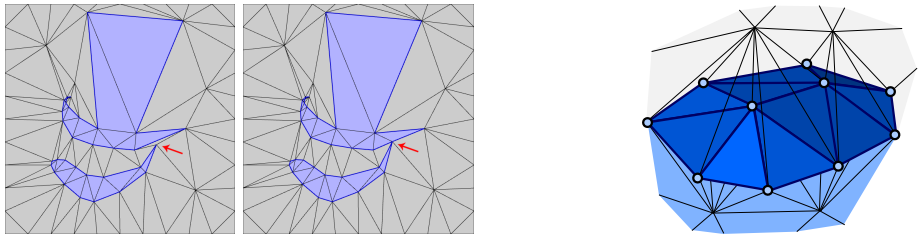

Figure 6.1: Interface representation in deformable simplicial complexes (2D on the left, 3D on the right). Exterior triangles (tetrahedra) are light gray, interior – blue. Simplices belonging to the interface (edges and vertices in 2D; faces, edges and vertices in 3D) are highlighted in dark blue. On the left, the red arrow indicates where topology changes take place. Note also the difference in scale between the largest and the smallest triangles.

the deforming velocity field ($\mathbf{u}$) directly to the interface points ($\mathbf{x}$):

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{u}(\mathbf{x}).$$

This approach leads to trouble once the topology of the interface changes. An efficient collision detection mechanism is needed to detect self-intersections of the interface, and once it happens, costly reparametrisation is needed, along with surgical cuts (as in [36], although in recent work by [13] this problem is mitigated by not allowing self-intersections). Those problems do not occur in Eulerian methods, such as the *level set method* (*LSM*, [75]). LSM represents a n-dimensional interface as the 0-level set of a $(n+1)$-dimensional function $f(x_1, \ldots, x_n, x_{n+1})$ (signed distance function is usually the choice), defined on the nodes of a regular grid. The evolution of the interface due to the velocity field $\mathbf{u}$ is then described by the following partial differential equation, also known as that *level set equation*:

$$\frac{\partial f}{\partial t} + \mathbf{u} \cdot \nabla f = 0.$$

This approach provides trivial and robust topological adaptivity. However, the LSM also exhibits several drawbacks: it is bound to a certain scale, it suffers from significant numerical diffusion for features near the sampling rate irrespective of discretization, it does not allow explicit interface representation and it relies on calculations in one dimension greater than the interface itself.

The work most directly related to ours is the method presented in [79]. The authors proposed a method which is based on a triangle mesh representation of the interface, but once the vertices have been moved, a restricted Delaunay tetrahedralization of the interface is performed. A test is performed on each of the new tetrahedra in order to label them as interior or exterior. If a vertex

is found to be shared only by identically labeled tetrahedra, it is removed. This method shares a number of advantages with our method. In particular, it can be extended to multi-phase simulations, and it suffers only little from numerical diffusion, but there is no detection of what happens between time steps. Arguably a small object could pass through a thin wall if the time step was not properly tuned, and the precise points where interface collisions occur are not detected. Lastly, it would be difficult to extend their method to do topology control which is simple with our approach.

In the *deformable simplicial complexes* (DSC), the interface is represented explicitly as a set of faces of simplices belonging to a simplicial complex one dimension higher. These simplices belong either to the object or the exterior. Simplices never straddle object boundaries. Thus, in 2D, the computational domain is divided into triangles, and the deforming interface is the set of line segments which divide interior triangles from exterior triangles. Similarly, in 3D, the interface is the set of triangles dividing interior tetrahedra from exterior tetrahedra. Both the 2D and 3D case are illustrated in Figure 6.1.

The interface deformation is performed by moving the vertices, and this means that the method preserves the advantages of the Lagrangian methods: It suffers from little numerical diffusion, and there is an explicit representation of the interface which, furthermore, does not change *gratuitously* between time steps. Moreover, the simplicial complex does not have to be regular meaning that we can allow details of significantly different scale in the same grid (c.f. Figure 6.1 left).

On the other hand, our approach also shares what we perceive as the biggest advantage of the Eulerian methods. Whenever the interface moves, the triangulation is updated to accommodate the change. If two different interface components collide, this change causes them to merge. Thus, topology is allowed to change transparently to the user—although with our method it would also be possible to disallow topological changes.

The DSC in described in detail, together with some of its other applications in [65].

6.4 Fluid Simulation

In DSC we attempt to keep the quality of the volume mesh high for the finite element computations, so it is natural to use it directly in the incompressible Euler equations solver. The fluid mass can be represented as the set of *interior*

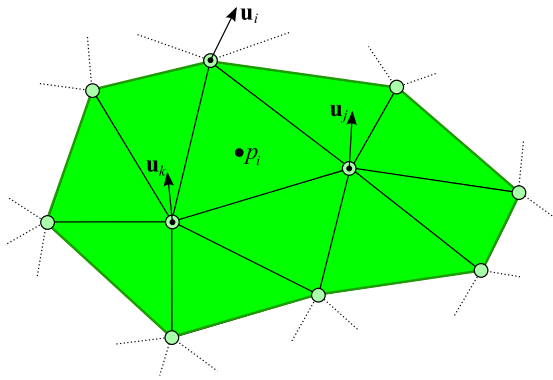

Figure 6.2: DSC setup for fluid simulation (observe this is analogical to a staggered mesh). The velocity values $\mathbf{u}$ are sampled at the vertices of the DSC mesh and the pressure values p are sampled per element (triangle in 2D, tetrahedron in 3D).

simplices, which can be treated as first order, *conforming* linear elements (meaning that velocity values in a vertex agree for each element sharing that vertex). These are subject to *locking* in the incompressible limit [24, 42, 103]. Locking means inability of a given finite element space to offer good approximate solutions, due to the fact that volume constraint on each tetrahedron may leave us with a solution space of very low dimension, or even an overconstrained problem (depending on the boundary conditions). This can manifest itself by, for example: only allowing globally affine divergence-free deformations of the fluid volume. However, locking can be avoided by using pressure stabilization [29], as presented in Section 6.4.4, in exchange for slighly violating the incompressibility constraint. Meanwhile, the simplicity of the linear elements facilitates easy implementation of advection and optimization-based implicit surface tension.

In such a setup, presented in the Figure 6.2, fluid velocity values are sampled in the vertices (both interface and interior ones) and pressure values are sampled in the centers of volume elements (triangles in the 2D case and tetrahedra in the 3D case). The velocity field is then defined as:

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^{N_V} \mathbf{u}_i \varphi_i(\mathbf{x}), \tag{6.1}$$

where N_V is the number of vertices in the mesh and φ_i is the linear interpolant (*hat* function defined on the star of vertex v_i).

Our method loosely follows the steps of a *fractional step method*, known from the regular-grid based fluid solvers [11].

6.4.1 Advection

In a Lagrangian setup (such as DSC) advection of the mesh vertices is trivial. Having vertex positions $\{\mathbf{v}_i^t\}_{i=1}^{N_V}$ and velocities $\{\mathbf{u}_i^t\}_{i=1}^{N_V}$ at the time-step t, one can compute the positions at the next time-step $t + \Delta t$ using simple forward Euler integration:

$$\mathbf{v}_i^{t+\Delta t} = \mathbf{v}_i^t + \mathbf{u}_i^t \Delta t.$$

One could also try to use a simple, Lagrangian approach in order to advect the velocity field. However, since we additionally perform smoothing on the mesh vertices, we have to use a slightly more complex, semi-Lagrangian method.

In order to advect the velocity field (or any other quantity sampled at the vertices) we interpolate or extrapolate the values from the previous time-step at the new vertex positions (see Figure 6.3). If the point $\mathbf{v}_i^{t+\Delta t}$ lies inside the fluid volume at the time-step t, we localize the element σ inside which it lies and compute the new velocity value as the linear combination of the velocities in the vertices of σ with barycentric coordinates of $\mathbf{v}_i^{t+\Delta t}$ as linear coefficients:

$$\mathbf{u}_i^{t+\Delta t} = \mathbf{u}^t(\mathbf{v}_i^{t+\Delta t}).$$

If $\mathbf{v}_i^{t+\Delta t}$ lies outside the fluid volume at the time-step t, we find its projection $\bar{\mathbf{v}}_i^{t+\Delta t}$ onto the interface and sample the velocity at this point:

$$\mathbf{u}_i^{t+\Delta t} = \mathbf{u}^t(\bar{\mathbf{v}}_i^{t+\Delta t}). \tag{6.2}$$

6.4.2 Enforcing Incompressibility

Incompressibility of the fluid yields that the divergence of the velocity field vanishes everywhere:

$$\nabla \cdot \mathbf{u} = 0.$$

In our setup (see equation 6.1):

$$\nabla \cdot \mathbf{u} = \sum_{i=1}^{N_V} \mathbf{u}_i \cdot \nabla \varphi_i.$$

The gradient $\nabla \varphi_i$ is constant over every element (triangle in 2D, tetrahedron in 3D). Let us denote it by:

$$\nabla \varphi_i \equiv \mathbf{d}_{j,i} \text{ over element } \sigma_j.$$

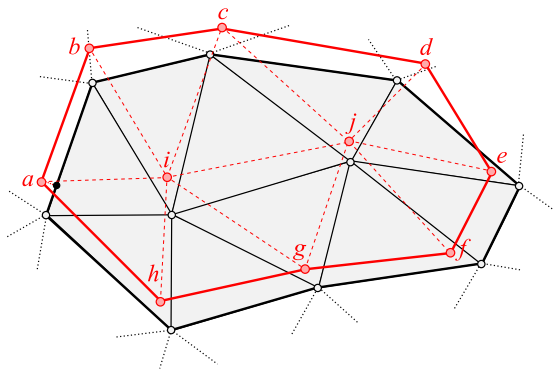

Figure 6.3: Advection of the velocity field. If the new vertex position is inside the old fluid volume (vertices e, f, g, h, i and j), we find its new velocity as the linear interpolation of old vertex velocities at this point. In order to find new velocity values at the vertices a, b, c and d, we find their projections onto the interface and sample the velocity there.

The incompressibility condition is then fulfilled iff:

$$\sum_{i=1}^{N_V} \mathbf{d}_{j,i} \cdot \mathbf{u}_i = 0 \quad \text{for } j = 1, \ldots, N_T,$$

where N_T is the number of elements (triangles in 2D, tetrahedra in 3D) in the mesh. The last equation can be written in matrix form:

$$\mathbf{D}\mathbf{u} = \mathbf{0},$$

where $\mathbf{u}$ is a size $d \cdot N_V$ (where the dimension $d = 2$ or 3) vector containing the coordinates of the vertex velocities and $\mathbf{D}$ is an $N_T \times d \cdot N_V$ sparse matrix.

To enforce incompressibility of the velocity field $\{\tilde{\mathbf{u}}_i\}_{i=1}^{N_V}$ after advection, we introduce a pressure field $\{p_i\}_{i=1}^{N_T}$, such that:

$$\mathbf{u} = \tilde{\mathbf{u}} - \mathbf{M}^{-1}\mathbf{D}^{\mathsf{T}}\mathbf{p}, \tag{6.3}$$

where $\mathbf{u}$ is divergence free, $\mathbf{p}$ is a size N_T vector containing the pressure values in each face and $\mathbf{M}$ is a size $d \cdot N_V \times d \cdot N_V$ diagonal mass matrix, with diagonal values:

$$\mathbf{M}_{d\cdot i-d+1,d\cdot i-d+1} = \ldots = \mathbf{M}_{d\cdot i,d\cdot i} = m_i,$$

for $i = 1, \ldots, N_V$, where:

$$m_i = \frac{1}{3}\rho \sum_{\sigma \in \mathrm{star}(v_i)} \mathrm{volume}(\sigma),$$

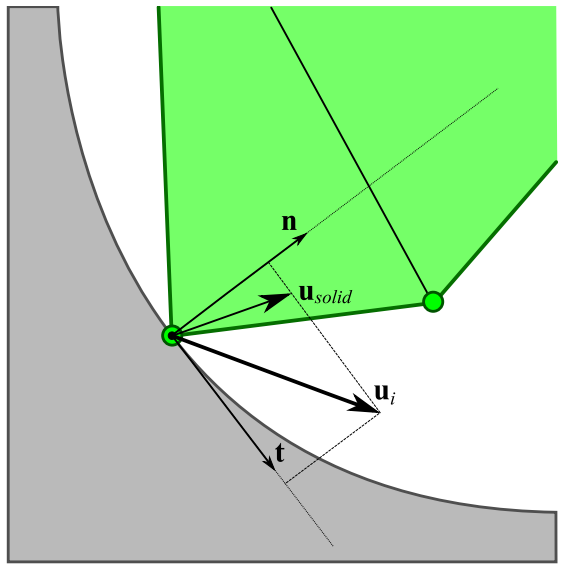

Figure 6.4: Collision of the fluid with the solid boundary. $\mathbf{n}$ is the normal and $\mathbf{t}$ is the tangent vector to the solid boundary at the point of collision. $\mathbf{u}_{solid}$ is the velocity of the solid boundary and $\mathbf{u}_i$ is constrained in the normal direction: $\langle \mathbf{u}_i, \mathbf{n} \rangle = \langle \mathbf{u}_{solid}, \mathbf{n} \rangle$.

with ρ the fluid density. The incompressibility condition yields:

$$\mathbf{Du} = \mathbf{D\tilde{u}} - \mathbf{DM}^{-1}\mathbf{D}^{\mathsf{T}}\mathbf{p} = \mathbf{0}.$$

Therefore:

$$\begin{aligned} \mathbf{DM}^{-1}\mathbf{D}^{\mathsf{T}}\mathbf{p} &= \mathbf{D\tilde{u}}, \\ \mathbf{Ap} &= \mathbf{b}, \end{aligned}$$

where $\mathbf{A} = \mathbf{DM}^{-1}\mathbf{D}^{\mathsf{T}}$ and $\mathbf{b} = \mathbf{D\tilde{u}}$. By solving this linear system, we can compute the pressure field and then, by using equation 6.3, the divergence-free velocity field $\mathbf{u}$.

Solid Boundaries Solid boundaries put extra constraints on vertex velocity values. If the vertex v_i is in contact with the solid (see Figure 6.4), we force the projection of the vertex's velocity onto the solid normal at the point of collision to match the projection of the solids own velocity onto its normal:

$$\langle \mathbf{u}_i, \mathbf{n}(\mathbf{p}_i) \rangle = \langle \mathbf{u}_{solid}, \mathbf{n}(\mathbf{p}_i) \rangle, \tag{6.4}$$

while the tangent coordinates of v_i remain unconstrained. In order to compute the new divergence-free velocity field $\{\mathbf{u}\}_{i=0}^{N_V}$ we first need to express the global velocity vector $\tilde{\mathbf{u}}$ and the matrix $\mathbf{D}$ in new coordinates ($\mathbf{n}$ and $\mathbf{t}$ in 2D or $\mathbf{n}$, $\mathbf{t}_1$ and $\mathbf{t}_2$ in 3D, whenever a vertex is in contact with the solid). Then we permute the rows of $\tilde{\mathbf{u}}$ and the columns of $\mathbf{D}$, so that:

$$\tilde{\mathbf{u}} = \left[\begin{array}{c} \tilde{\mathbf{u}}_f \\ \hline \tilde{\mathbf{u}}_c \end{array}\right], \qquad \mathbf{D} = \left[\begin{array}{c|c} \mathbf{D}_f & \mathbf{D}_c \end{array}\right] \tag{6.5}$$

where $\tilde{\mathbf{u}}_f$ contains the free and $\tilde{\mathbf{u}}_c$ the constrained coordinates of $\tilde{\mathbf{u}}$. The incompressibility condition 6.3 can be then written as:

$$\begin{aligned} \mathbf{D}\mathbf{u} &= \mathbf{0}, \\ \left[\begin{array}{c|c} \mathbf{D}_f & \mathbf{D}_c \end{array}\right] \left[\begin{array}{c} \mathbf{u}_f \\ \hline \mathbf{u}_c \end{array}\right] &= \mathbf{0}, \\ \mathbf{D}_f\mathbf{u}_f + \mathbf{D}_c\mathbf{u}_c &= \mathbf{0}, \end{aligned}$$

but since $\mathbf{u}_c = \tilde{\mathbf{u}}_c$ (as they are forced to match the velocity of the solid projected onto the solid normal), we obtain:

$$\mathbf{D}_f\mathbf{u}_f = -\mathbf{D}_c\tilde{\mathbf{u}}_c. \tag{6.6}$$

In order to ensure incompressibility of the velocity field $\mathbf{u}$, we again introduce a pressure field $\mathbf{p}$:

$$\mathbf{u}_f = \tilde{\mathbf{u}}_f - \mathbf{M}_f^{-1}\mathbf{D}_f^\mathsf{T}\mathbf{p}.$$

Multiplying both sides by $\mathbf{D}_f$ and using eq. 6.6 gives:

$$\begin{aligned} \mathbf{D}_f\mathbf{u}_f &= \mathbf{D}_f\tilde{\mathbf{u}}_f - \mathbf{D}_f\mathbf{M}_f^{-1}\mathbf{D}_f^\mathsf{T}\mathbf{p}, \\ -\mathbf{D}_c\tilde{\mathbf{u}}_c &= \mathbf{D}_f\tilde{\mathbf{u}}_f - \mathbf{D}_f\mathbf{M}_f^{-1}\mathbf{D}_f^\mathsf{T}\mathbf{p}, \end{aligned}$$

which can be used to evaluate $\mathbf{p}$ by solving a linear system:

$$\begin{aligned} \mathbf{D}_f\mathbf{M}_f^{-1}\mathbf{D}_f^\mathsf{T}\mathbf{p} &= \mathbf{D}_f\tilde{\mathbf{u}}_f + \mathbf{D}_c\tilde{\mathbf{u}}_c, \\ \mathbf{A}_f\mathbf{p} &= \mathbf{D}_f\tilde{\mathbf{u}}_f + \mathbf{D}_c\tilde{\mathbf{u}}_c. \end{aligned}$$

Gravity Including gravity in our fluid dynamics solver is trivial and can be performed by adding $\mathbf{g}\Delta t$ to the velocity field in every time step, where $\mathbf{g}$ is the gravitational acceleration.

6.4.3 Optimization Based Approach

Surface Tension In order to make our fluid simulation more plausible we include *surface tension*. Surface tension is derived from *surface energy* U_γ defined

as:

$$U_\gamma = \gamma A,$$

where γ is the surface energy density (material constant) and A is the free surface area. Surface tension forces alone yield a highly divergent velocity field and our experiments have shown that integrating them before enforcing incompressibility step can give very poor results. Instead, we fully couple them with incompressibility by solving the following optimization problem:

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}(\mathbf{u}-\tilde{\mathbf{u}})^\mathsf{T}\mathbf{M}(\mathbf{u}-\tilde{\mathbf{u}}) + U_\gamma(\mathbf{x}+\mathbf{u}\Delta t), \\ \text{subject to} & \mathbf{D}\mathbf{u} = \mathbf{0}, \end{array} \tag{6.7}$$

which is consistent as the first-order optimality conditions for the optimality of its solution is the backward Euler step:

$$\mathbf{u} = \tilde{\mathbf{u}} - \Delta t \mathbf{M}^{-1}\nabla U_\gamma(\mathbf{u}) - \mathbf{M}^{-1}\mathbf{D}^\mathsf{T}\mathbf{p},$$

where the pressure values $\mathbf{p}$ play the role of Lagrange multipliers (see that for $\gamma = 0$ this is identical with eq. 6.3). In our work so far we only use first-order approximation of the surface energy function:

$$\begin{array}{rcl} U_\gamma(\mathbf{x}+\mathbf{u}\Delta t) & = & \gamma A(\mathbf{x}+\mathbf{u}\Delta t) \approx \\ & \approx & \gamma A(\mathbf{x}) + \gamma \Delta t \mathbf{k}^\mathsf{T}\mathbf{u}, \end{array}$$

where $\mathbf{k}^\mathsf{T}$ is the area gradient ∇A. Substituting this into optimization problem 6.7 and dropping constant terms leads to a simple quadratic programming [73] problem with linear equality constraints:

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}\mathbf{u}^\mathsf{T}\mathbf{M}\mathbf{u} + (-\mathbf{M}\tilde{\mathbf{u}} + \gamma\Delta t\mathbf{k})^\mathsf{T}\mathbf{u}, \\ \text{subject to} & \mathbf{D}\mathbf{u} = \mathbf{0}. \end{array}$$

In this fashion we avoid having to explicitly estimate surface curvature, automatically conserve linear and angular momentum by virtue of translation and rotation-independence of the objective function and naturally capture minimum-surface-area equilibrium. We do, however, realize that this setup does not allow for non-linear surface phenomena in our simulations.

Solid Boundaries Incorporating solid boundaries into the new setting is relatively straightforward. However, one has to take into account the fact that the surface energy density for the fluid-air surface γ is usually different from the surface energy density for the fluid-solid surface $\alpha_1\gamma$ and from the surface energy density for the air-solid surface $\alpha_2\gamma$. Hence, we multiply the area of the solid-liquid contact surface by $\alpha_1 - \alpha_2$. Using the notation from the previous

section, the zero-divergence constraint is decribed by the eq. 6.6. Also, the surface energy has to be expressed in the new variables:

$$\begin{aligned} U_\gamma(\mathbf{x}+\mathbf{u}\Delta t) &= \gamma A(\mathbf{x}+\mathbf{u}\Delta t) \approx \\ &\approx \gamma A(\mathbf{x}) + \gamma\Delta t\mathbf{k}_f^\mathsf{T}\mathbf{u}_f + \gamma\Delta t\mathbf{k}_c^\mathsf{T}\mathbf{u}_c, \end{aligned}$$

where $\mathbf{k}_f$ is a vector containing those coordinates of the area gradient ∇A, which correspond to the free coordinates of $\mathbf{u}$ (put together in a vector $\mathbf{u}_f$) and $\mathbf{k}_c$ contains those cordinates of ∇A, which correspond to the constrained coordinates of $\mathbf{u}$ (put together in a vector $\mathbf{u}_c$). Finally, after dropping constant terms and terms depending only on $\mathbf{u}_c$, we can state our optimization problem in the following form:

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}\mathbf{u}_f^\mathsf{T}\mathbf{M}_f\mathbf{u}_f + (-\mathbf{M}_f\tilde{\mathbf{u}}_f + \gamma\Delta t\mathbf{k}_f)^\mathsf{T}\mathbf{u}_f, \\ \text{subject to} & \mathbf{D}_f\mathbf{u}_f = -\mathbf{D}_c\tilde{\mathbf{u}}_c. \end{array} \tag{6.8}$$

Solution For sake of simplicity, let us rewrite the optimization problem 6.8 as:

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}\mathbf{u}_f^\mathsf{T}\mathbf{M}_f\mathbf{u}_f - \mathbf{b}\mathbf{u}_f, \\ \text{subject to} & \mathbf{D}_f\mathbf{u}_f = \mathbf{c}, \end{array}$$

where $\mathbf{b} = \mathbf{M}_f\tilde{\mathbf{u}}_f - \gamma\Delta t\mathbf{k}_f$ and $\mathbf{c} = -\mathbf{D}_c\tilde{\mathbf{u}}_c$. The solution of this optimization problem can be found by solving the following linear equation [73]:

$$\begin{bmatrix} \mathbf{M}_f & \mathbf{D}_f^\mathsf{T} \\ \mathbf{D}_f & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}. \tag{6.9}$$

We are doing this by applying Schur complement method for solving linear systems with block matrices [101]. It produces the following linear equation:

$$-\mathbf{D}_f\mathbf{M}_f^{-1}\mathbf{D}_f^\mathsf{T}\mathbf{p} = \mathbf{c} - \mathbf{D}_f\mathbf{M}_f^{-1}\mathbf{b}.$$

Then, having found $\mathbf{p}$:

$$\mathbf{u}_f = \mathbf{M}_f^{-1}\mathbf{b} - \mathbf{M}_f^{-1}\mathbf{D}_f^\mathsf{T}\mathbf{p}.$$

This way, we are only required to solve a linear equation with a size $N_T \times N_T$ matrix (instead of size $d\cdot N_V + N_T \times d\cdot N_V + N_T$ original problem), as computing the inverse of the diagonal matrix $\mathbf{M}_f$ is trivial.

6.4.4 Pressure Stabilization

As we previously mentioned, presented finite element setup is subject to *locking*. This problem can be solved by adding a stabilization term $\mathbf{S}$ (size $N_T \times N_T$) to

the linear equation 6.9:

$$\begin{bmatrix} \mathbf{M}_f & \mathbf{D}_f^\mathsf{T} \\ \mathbf{D}_f & -\mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}. \tag{6.10}$$

such that:

$$\mathbf{S}_{ij} = \begin{cases} -\delta \cdot a_{ij} & \text{if } i \neq j \\ \delta \cdot \sum_{k \neq i} a_{ik} & \text{if } i = j \end{cases} \tag{6.11}$$

where δ is a positive stabilization parameter and a_{ij} is the area of the face shared by tetrahedra i and j, if they do have a common face, or otherwise 0. Stabilization term of this form acts like Laplacian smoothing of the pressure field in exchange for slightly violating the incompressibility constraint.

In order to solve equation 6.10, we again apply Schur complement method and solve the following equation:

$$-(\mathbf{S} + \mathbf{D}_f \mathbf{M}_f^{-1} \mathbf{D}_f^\mathsf{T})\mathbf{p} = \mathbf{c} - \mathbf{D}_f \mathbf{M}_f^{-1} \mathbf{b}.$$

Then, having found $\mathbf{p}$:

$$\mathbf{u}_f = \mathbf{M}_f^{-1} \mathbf{b} - \mathbf{M}_f^{-1} \mathbf{D}_f^\mathsf{T} \mathbf{p}.$$

Even though the velocity field $\mathbf{u}$ computed this way is not divergence-free in each individual tetrahedron, it is still *globally* volume preserving for a stabilization term $\mathbf{S}$ defined as above. It is easy to notice that the form of stabilization term 6.11 yields that the sum of all coordinates of the vector $\mathbf{Sp}$ equals 0. Since:

$$\begin{aligned} \mathbf{Sp} &= \mathbf{D}_f \mathbf{u}_f - \mathbf{c} \\ &= \mathbf{D}_f \mathbf{u}_f + \mathbf{D}_c \tilde{\mathbf{u}}_c \\ &= \mathbf{D}_f \mathbf{u}_f + \mathbf{D}_c \mathbf{u}_c = \mathbf{D}\mathbf{u}, \end{aligned}$$

that means that the integral of the divergence of the velocity field over the fluid volume equals 0, hence it preserves global volume.

6.4.5 Volume Loss Compensation

If the volume loss due to the truncation errors is visually noticeable, one can compensate for that by adding a constant term:

$$\tau \cdot \frac{V_0 - V_c}{\Delta t \cdot V_c}$$

to the desired divergence $\mathbf{c}$, where V_0 is the original volume of the fluid, V_c is the current volume and τ is a relaxation parameter ($\tau = 0.5$ being a good choice).

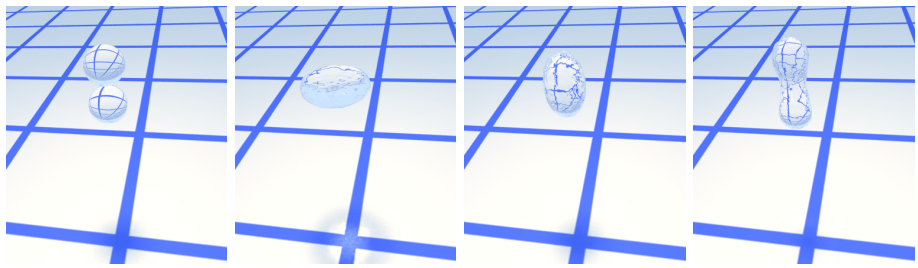

Figure 6.5: Head-on collision of two water droplets in 0-gravity conditions.

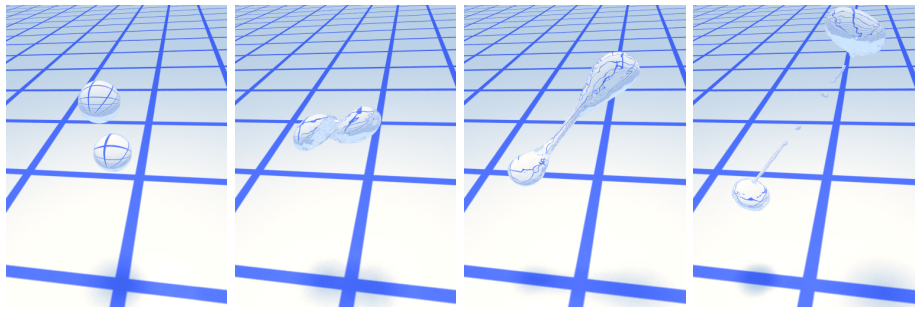

Figure 6.6: Oblique collision of two water droplets in 0-gravity conditions.

6.5 Tests and Results

Stationary Volume of Fluid We first tested our solver on a regular sphere model obtained by subdiving an icosahedron 4 times using $\sqrt{3}$-subdivision scheme [53] and reprojecting the vertices onto a sphere. The initial velocity of all vertices are set to zero and there is no gravity – the only forces in this setup are due to surface tension and incompressibility.

After 10000 iterations the changes in volume and surface area are below floating point truncation error and there is no visible displacement of the mesh vertices, as expected for this symmetric situation.

Droplets Colliding Our next test involves two water droplets in 0-gravity conditions. In the first experiment (see Figure 6.5) they collide head-on, and in the second (see Figure 6.6) they collide obliquely. In the first case droplets merge and the resulting volume begins to oscillate between flattened and elongated shape, according to the energy conservation principle. In the second case

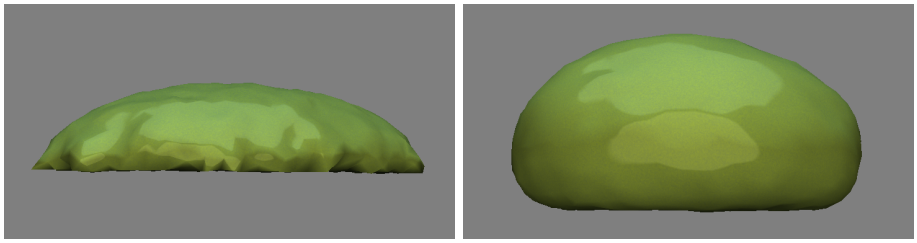

Figure 6.7: Shape aquired by a droplet of liquid put on a flat surface. On the left-hand side: $\alpha_1 - \alpha_2 = -0.75$, and indeed the contact angle $\theta < 90°$. On the right-hand side: $\alpha_1 - \alpha_2 = 0.75$, and indeed the contact angle $\theta > 90°$. Note that the pond shape is more irregular when $\theta < 90°$. This is the case also in the physical world, e.g.: ponds of water on the glass usually acquire quite irregular shapes, while ponds of mercury on the glass are usually round. This is due to the fact, that increasing the contact surface between water and glass decreases the total energy of the system, unlike in the latter case.

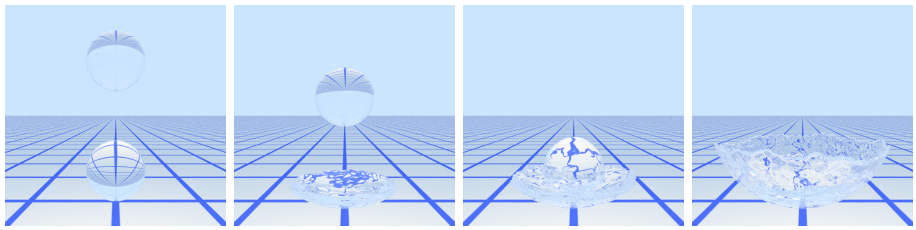

Figure 6.8: Two drops of water splashing inside a solid sphere.

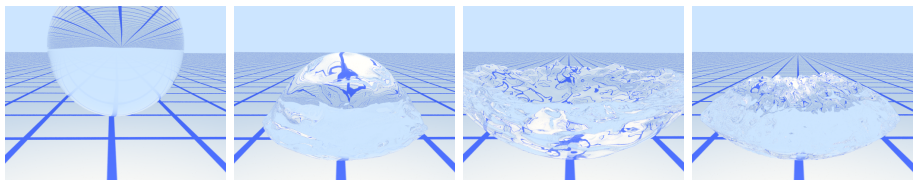

Figure 6.9: Large drop of water splashing inside a solid sphere. $\alpha_1 - \alpha_2 = 0$ (corresponding to contact angle $\theta = 90°$, characteristic for e.g. water on silver), surface energy density exagerrated.

droplets first merge, but as two big fractions of volume keep moving in original directions they detach, leaving a trace of small droplets, which soon start to oscillating around the spherical equilibrium.

Fluid-Solid Interaction In the first test, we examined the behavior of a droplet of fluid put on a flat surface, subject to gravitational force. Let γ be the surface energy density for the fluid-air interface, $\alpha_1\gamma$ – for the fluid-solid interface and $\alpha_2\gamma$ – for the air-solid interface. Then the contact angle θ between the fluid-air surface and the solid surface equals:

$$\cos\theta = -(\alpha_1 - \alpha_2).$$

A concave *meniscus* has contact angle less than 90° (e.g. water on glass) and a convex meniscus has contact angle greater than 90° (e.g. water on paraffin wax or mercury on glass). We ran tests for $\alpha_1 - \alpha_2 = 0.75$ and $\alpha_1 - \alpha_2 = -0.75$. The results, presented in Figure 6.7 demonstrate physical soundness of our method.

We also tested our fluid simulation in scenarios involving curved solid boundaries. The results, presented in Figures 6.8 and 6.9, demonstrate that the curved boundaries are handled correctly in our setup.

Performance In all of our experiments the number of tetrahedra was on the order of 10000. Simulation time ranged from about 10 to 30 iterations per minute (on 64-bit Intel® Xeon® CPU W5590 @ 3.33 GHz, 6 GB RAM) – not including rendering which was done in a subsequent step.

6.6 Conclusions and Future Work

In this paper, we have demonstrated the feasibility of fluid simulation in a framework where the computational grid evolves over time, maintaining the fluid interface as a subcomplex of the tetrahedral grid. This is in contrast to the few examples of previous work which used unstructured grids. In these methods, the computational grid is either fixed or, essentially, rebuilt every time step.

Because of this, we have an explicit fluid surface representation in the form of a triangle mesh which is also not rebuilt every time step since it is a subcomplex of the tetrahedral grid. From this we derive one of the big advantages of the method, namely that we can easily formulate surface energy in terms of the surface geometry.

Since, arguably, this method is qualitatively different from previous unstructured mesh based methods for fluid simulation, it is unsurprising that there is room for future improvement.

It is clear from our screenshots and animations that the fluid surface is quite rough in some cases. This is due to the lack of a viscosity term which should be straightforward to add and make it faster to reach the equilibrium state. Note also that in grid based methods, unintentional viscosity is quite common due to numerical diffusion. In fact, one of the strengths of our method is that there is very little numerical diffusion since we only change the mesh when parts of the surface collide (to change topology of the fluid volume) or when we need to remove poor quality tetrahedra.

One of the direct and straight-forward short term goals is investigating the influence of using second-order surface energy approximation, which can easily be included in the existing framework. We are also planning to try using Sequential Quadratic Programming in order to investigate the influence of higher-order terms (although we believe this might require using more elaborate mesh refinement schemes).

Furthermore, our method can be extended by adding viscosity, allowing compressible fluids and multiple phases (supported naturally by the DSC). Ultimately, we would like to include solid (rigid, ellastic and deformable) objects in an unified physis simulation setup.

Acknowledgements

We would like to thank Jeppe Revall Frisvad (DTU Informatics) for providing us with the water rendering software.

Chapter 7

Conclusions and outlook

We have presented *deformable simplicial complexes*, a novel method for deformable interface tracking, significantly different from existing, state-of-the-art methods. We have shown that the DSC can address some of the shortcomings of other topology-adaptive deformable models: difficulties preserving sharp details, poor space adaptivity, troublesome topology control. Moreover, having explicit representations of both the interface and the space turned out to give extra benefits: The DSC mesh can be used as a computational grid for the finite element method; its structure allows the domain to have topology other than a disk and it does not have to be Euclidean.

Having said this, we are well aware that the deformable simplicial complexes have some drawbacks and limitations, one being time complexity. The method, although scalable, yields significant time overhead compared to the level set method and so, in its current form, it cannot be applied in real-time applications. It also seems that unlike the level set method, the DSC might be rather challenging. This is due to the fact that the DSC is much more complex in its use of various methods to improve the quality of the underlying mesh. However, luckily most of the DSC components are local in nature and hence they could be parallelized. We are planning to inspect this problem in the future.

In this project we concentrated on developing the fundametal method (including the low-level framework) and exploring the applications. However, there is still

work to be done. We would like to perform more rigorous, systematic analysis of the properties of the deformable simplicial complexes, in order to better understand what effect the individual components of the method have on its performance. This would be the natural, next step of the research process concerning the DSC.

As long as we were trying to optimize the performance of the method while working on the first implementation, we believe there is still room for improvement. Some speed-up could be gained by further modifying the incidence simplicial data structure, which was used to represent the connectivity of the mesh. We could make the simplicial complex traversal faster by utilizing its manifoldness and storing different boundary and co-boundary relations. It has not been done yet, because it was of higher priority to start investigating the properties of the method as soon as we had a stable, applicable tetrahedral mesh implementation. Another way of performance optimization would be through localizing the mesh improvement operations to the regions where the interface deformation is significant.

Each of the applications we have worked on gave promising results and could be further developed. There is room for more experimentation with our fluid solver. One could easily incorporate more accurate approximation of the surface energy; it would also be interesting to have more than just 2 phases in the setup. In longer run, it would be tempting to try implementing a unified physics-based simulation framework where one could simulate fluids alongside rigid and ellastic solid bodies. The DSC seem to be very well suited for this kind of application, since it naturally supports multiple phases, provides an immediate collision detection mechanism as well as a computational grid.

Our cut locus construction algorithm could also be extended—it would be particularily tempting to implement it on top of 3D deformable simplicial complexes. Little is known about cut loci on 3-manifolds, so such an application could generate significant new knowledge. It would also be interesting to apply the existing, 2D algorithm to real life problems, such as forest fire modelling.

Our simple point cloud reconstruction method, implemented within just a couple of weeks turned out to be surprisingly effective: It is robustly space-adaptive and easily captures sharp details. We would like to continue developing this method, and try applying it to more challenging point-cloud reconstruction problems like, for example, reconstruction from sparse sets of points.

Finally, we would like to try applying the DSC to other problems, especially in creating topology adaptive shape models (for example, cranial growth model in babies with congenital disorders) and to structural optimization.

Bibliography

[1] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 528–537. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1997.

[2] G. Arfken. *Mathematical Methods for Physicists*. Academic Press, Orlando, FL, 3rd ed. edition, 1985.

[3] Kendall Atkinson and Weimin Han. *Theoretical Numerical Analysis: A Functional Analysis Framework*. Springer, 2 edition, 2005.

[4] Jongmin Baek, Anand Deopurkar, and Katherine Redfield. Finding geodesics on surfaces. Unpublished manuscript, 2007.

[5] J. A. Bærentzen. *Introduction to GEL*, 2010.

[6] Jakob Andreas Bærentzen and Niels Jørgen Christensen. Interactive modelling of shapes using the level-set method. *International Journal of Shaping Modeling*, 8(2):79–97, 2002.

[7] M. P. Bendsøe and O. Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer, 2003.

[8] Stephan Bischoff and Leif Kobbelt. Topologically correct extraction of the cortical surface of a brain using level-set methods. In *In Proceedings of BVM 2004*, pages 50–54, 2004.

[9] CGAL Editorial Board. *CGAL-3.2 User and Reference Manual*, 2006.

[10] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh - a generic and efficient polygon mesh data structure, 2002.

[11] Robert Bridson. *Fluid Simulation.* A. K. Peters, Ltd., Natick, MA, USA, 2008.

[12] Tyson Brochu, Christopher Batty, and Robert Bridson. Matching fluid simulation elements to surface geometry and topology. In *ACM SIGGRAPH 2010 papers*, page XX. ACM, 2010.

[13] Tyson Brochu and Robert Bridson. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493, 2009.

[14] T. Chan and L. Vese. A level set algorithm for minimizing the mumford-shah functional in image processing. In *Proceedings of the 1st IEEE Workshop on "Variational and Level Set Methods in Computer Vision"*, pages 161–168, 2001.

[15] L. Chen and J. Xu. Optimal delaunay triangulation. *J. Comp. Math*, 22:299–308, 2004.

[16] Nuttapong Chentanez, Bryan E. Feldman, François Labelle, James F. O'Brien, and Jonathan R. Shewchuk. Liquid simulation on lattice-based tetrahedral meshes. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 219–228, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[17] Nuttapong Chentanez, Tolga G. Goktekin, Bryan E. Feldman, and James F. O'Brien. Simultaneous coupling of fluids and deformable bodies. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 83–89, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[18] L.P. Chew. Guaranteed-quality delaunay meshing in 3d (short version). In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 391–393. ACM New York, NY, USA, 1997.

[19] B. Cutler, J. Dorsey, and L. McMillan. Simplification and improvement of tetrahedral models for simulation. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 93–102, New York, NY, USA, 2004. ACM.

[20] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1997.

[21] Hugues L. de Cougny and Mark S. Shephard. Refinement, derefinement and optimization of tetrahedral geometric triangulations in three dimensions. Unpublished manuscript, 1995.

[22] Q. Du and D. Wang. Tetrahedral mesh generation and optimization based on centroidal voronoi tessellations. *Int. J. Numer. Meth. Eng*, 56:1355–1373, 2002.

[23] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1):4, 2007.

[24] E. English and R. Bridson. Animating developable surfaces using nonconforming elements. In *ACM SIGGRAPH 2008 papers*, page 66. ACM, 2008.

[25] Douglas Enright, Steve Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics*, 21(3):736–44, 2002.

[26] Lawrence Evans. *Partial Differential Equations*. American Mathematical Society, 1998.

[27] Bryan E. Feldman, James F. O'Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 904–909, New York, NY, USA, 2005. ACM.

[28] Bryan E. Feldman, James F. O'Brien, Bryan M. Klingner, and Tolga G. Goktekin. Fluids in deforming meshes. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 255–259, New York, NY, USA, 2005. ACM.

[29] Joel H. Ferziger and Milovan Peric. *Computational Methods for Fluid Dynamics*. Springer, 3rd edition, 2002.

[30] L. De Floriani, A. Hui, D. Panozzo, and D. Canino. A dimension-independent data structure for simplicial complexes. In *Proceedings of the 19th International Meshing Roundtable*, 2010.

[31] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483, 1996.

[32] Lori A. Freitag. On combining laplacian and optimization-based mesh smoothing techniques. In *In Trends in Unstructured Mesh Generation*, pages 37–43, 1997.

[33] Lori A. Freitag, Mark Jones, and Paul Plassmann. An efficient parallel algorithm for mesh smoothing. In *Proceedings of the Fourth International Meshing Roundtable*, pages 103–112, 1995.

[34] Lori A. Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40:3979–4002, 1997.

[35] J. Glasa and L. Halada. On elliptical model for forest fire spread modeling and simulation. *Mathematics and Computers in Simulation*, 78:76–88, 2008.

[36] James Glimm, John W. Grove, Xiao Lin Li, Keh-Ming Shyue, Yanni Zeng, and Qiang Zhang. Three dimensional front tracking. *SIAM J. Sci. Comp*, 19:703–727, 1995.

[37] Frederik Gottlieb. Deformable simplicial complexes. Master thesis, 2008. Despite the title, this MSc thesis deals almost exclusively with the data storage kernel of the tetrahedral mesh data structure.

[38] J. Gravesen, S. Markvorsen, R. Sinclair, and M. Tanaka. The cut locus of a torus of revolution. *Asian Journal of Mathematics*, 9(1):103–120, 2005.

[39] Leonidas J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, April 1985.

[40] Mads F. Hansen, Jakob A. Bærentzen, and Rasmus Larsen. Generating quality tetrahedral meshes from binary volumes. In *Proceedings of VISAPP 2009*, 2009.

[41] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *ACM SIGGRAPH 1993 Conference Proceedings*, pages 19–26, 1993.

[42] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Transactions on Graphics (TOG)*, 25(3):811, 2006.

[43] J. Itoh and R. Sinclair. Thaw: A tool for approximating cut loci on a triangulation of a surface. *Experimental Mathematics*, 13(3):309–325, 2004.

[44] Xiangmin Jiao. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys.*, 220(2):612–625, 2007.

[45] Peter Stanley Jørgensen, Karin Vels Hansen, Rasmus Larsen, and Jacob R. Bowen. High accuracy interface characterization of three phase material systems in three dimensions. *Journal of Power Sources*, 195(24):8168–8176, 2010.

[46] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[47] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[48] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proc. of the Natl. Acad. Sciences USA*, 95:8431–8435, 1998.

[49] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[50] G.T. Klincsek. Minimal triangulations of polygonal domains. *Annals of Discrete Mathematics*, 9:121–123, 1980.

[51] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. Fluid animation with dynamic meshes. *ACM Trans. Graph.*, 25(3):820–825, 2006.

[52] Bryan M. Klingner and Jonathan R. Shewchuk. Agressive tetrahedral mesh improvement. In *Proceedings of the 16th International Meshing Roundtable*, pages 3–23, October 2007.

[53] Leif Kobbelt. $\sqrt{3}$-subdivision. In *Proceedings of the 27th annual conference on computer graphics and interactive techniques*, 2000.

[54] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics*, 26(3):57, 2007.

[55] J.O. Lachaud and A. Montanvert. Deformable meshes with automated topology changes for coarse-to-fine three-dimensional surface extraction. *Medical Image Analysis*, 3(2):187–207, 1999.

[56] E. Laporte and P. Le Tallec. *Numerical Methods in Sensitivity Analysis and Shape Optimization*. Birkhäuser, 2003.

[57] John M. Lee. *Introduction to Topological Manifolds*. Graduate Texts in Mathematics. Springer, 2000.

[58] Randall J. LeVeque. *Numerical Methods for Conservation Laws*. Birkäuser, 1992.

[59] Jianfei Liu and Shuli Sun. Small polyhedron reconnection: A new way to eliminate poorly-shaped tetrahedra. In *Proceedings of the 15th International Meshing Roundtable*, pages 241–257, 2006.

[60] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4), 1987.

[61] T. Maekawa. Computation of shortest paths on free-form parametric surfaces. *J. of Mech. Design*, 118(4):499–508, 1996.

[62] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, 1988.

[63] A. Marquina and S. Osher. Explicit algorithms for a new time dependent model based on level set motion for nonlinear deblurring and noise removal. *SIAM J. Sci. Comput.*, pages 387–405, 2000.

[64] T. McInerney and D. Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, 2000.

[65] M. K. Misztal and J. A. Bærentzen. Deformable simplicial complexes. Unpublished manuscript, 2010.

[66] M. K. Misztal, J. A. Bærentzen, and S. Markvorsen. Cut locus construction using deformable simplicial complexes. Unpublished manuscript, 2010.

[67] M. K. Misztal, R. Bridson, K. Erleben, J. A. Bærentzen, and F. Anton. Optimization-based fluid simulation on unstructured meshes. In *VRIPHYS 2010: Proceedings of the 7th Workshop on Virtual Reality Interaction and Physical Simulation*, 2010.

[68] Marek K. Misztal, J. Andreas Bærentzen, François Anton, and Kenny Erleben. Tetrahedral mesh improvement using multi-face retriangulation. In *Proceedings of the 18th International Meshing Roundtable*, pages 539–556, October 2009.

[69] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *Proc. International Meshing Roundtable*, 2003.

[70] P. M oller and P. Hansbo. On advancing front mesh generation in three dimensions. *International Journal for Numerical Methods in Engineering*, 38(21):3551–3569, 1995.

[71] V. Natarajan and H. Edelsbrunner. Simplication of three-dimensional density maps. *IEEE Transactions on Visualization and Computer Graphics*, 10:587–597, 2004.

[72] Timothy S. Newman and Hong Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879, 2006.

[73] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 2 edition, 2006.

[74] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Computational Physics*, 79:12–49, 1988.

[75] Stanley J. Osher and Ronald P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 1 edition, October 2002.

[76] V. N. Parthasarathy, C. M. Graichen, and A. F. Hathaway. A comparison of tetrahedron quality measures. *Finite Elements in Analysis and Design*, 15(3):255–261, 1994.

[77] X. Pennec. Toward a generic framework for recognition based on uncertain geometric features. *Journal of Computer Vision Research*, 1(2):58–87, 1998.

[78] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[79] Jean-Philippe Pons and Jean-Daniel Boissonnat. Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2007.

[80] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[81] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, pages 259–268, 1992.

[82] T. Sakai. *Riemannian Geometry*, volume 149 of *Translations of Mathematical Monographs*. American Mathematical Society, 1996.

[83] J. A. Sethian. *Level Set Methods and Fast Marching Methods. Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

[84] Ojaswa Sharma, Qin Zhang, François Anton, and Chandrajit Bajaj. Multi-domain, higher order level set scheme for 3d image segmentation on the gpu. In *The 23rd IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

[85] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004*, pages 896–904. ACM Press, August 2004.

[86] Jonathan R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 86–95. ACM New York, NY, USA, 1998.

[87] Jonathan R. Shewchuk. Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. Unpublished manuscript, 2002.

[88] Jonathan R. Shewchuk. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures. Unpublished manuscript, 2002.

[89] Hang Si. Tetgen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, v1.3 user's manual. Technical report, WIAS, 2004.

[90] R. Sinclair and M. Tanaka. Loki: Software for computing cut loci. *Experimental Mathematics*, 11(1):1–25, 2002.

[91] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[92] Nils Thürey, Chris Wojtan, Markus Gross, and Greg Turk. A multiscale approach to mesh-based surface tension flows. In *SIGGRAPH '10: ACM SIGGRAPH 2010 papers*, pages 1–10, New York, NY, USA, 2010. ACM.

[93] Greg Turk and James F. O'brien. Modelling with implicit surfaces that interpolate. *ACM Trans. Graph.*, 21(4):855–873, 2002.

[94] Michael Yu Wang, Xiaoming Wang, and Dongming Guo. A level set method for structural topology optimization. *Comput. Methods Appl. Mech. Engrg.*, 192:227–246, 2003.

[95] Jeremy D. Wendt, William Baxter, Ipek Oguz, and Ming C. Lin. Finite volume flow simulations on arbitrary domains. *Graph. Models*, 69(1):19–32, 2007.

[96] C. Wojtan, N. Thürey, M. Gross, and G. Turk. Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 papers*, page 76. ACM, 2009.

[97] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Physics-inspired topology changes for thin fluid features. In *SIGGRAPH '10: ACM SIGGRAPH 2010 papers*, pages 1–8, New York, NY, USA, 2010. ACM.

[98] Chris Wojtan and Greg Turk. Fast viscoelastic behavior with thin features. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–8, New York, NY, USA, 2008. ACM.

[99] A. Zaharescu, E. Boyer, and R. Horaud. Transformesh: a topology-adaptive mesh-based approach to surface evolution. *Computer Vision–ACCV 2007*, pages 166–175, 2007.

[100] Andrei Zaharescu, Edmond Boyer, and Radu P. Horaud. Topology-adaptive mesh deformation for surface evolution, morphing, and multi-view reconstruction. Technical Report RR-7136, INRIA Grenoble Rhone-Alpes, December 2009.

[101] Fuzhen Zhang. *The Schur Complement and Its Applications*. Springer, 2005.

[102] H.-K. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit and nonparametric shape reconstruction from unorganized data using a variational level set method. *Comput. Vision and Image Understanding*, 80:295–314, 2000.

[103] O. C. Zienkiewicz, R. L. Taylor, and R. L. Taylor. *The finite element method for solid and structural mechanics*. Butterworth-Heinemann, 2005.